

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

全面系统论述了HTML5 App开发技术及可二次开发的案例
融合一线教学与科研经验的经典著作

实例源码 | 教学课件 | 练习素材

HTML5 App Design and Development

HTML5 App

应用开发教程

黄波 张小华 黄平 王彩◎编著



清华大学出版社

内容简介

本书以HTML5应用开发为主线，从HTML5的起源、发展、应用、开发、测试、部署等方面进行了全面的介绍。本书不仅是一本HTML5应用开发的入门书籍，也是一本HTML5应用开发的进阶书籍。本书适合从事HTML5应用开发的开发人员阅读，也适合从事HTML5应用开发的管理人员阅读。



为什么要写这本书

近年来移动互联网的发展十分迅猛，HTML5应用开发也受到了世界各国的广泛关注和大力支持。大量的投资涌入HTML5应用开发领域，HTML5应用开发已经成为移动互联网领域的热门话题。目前主流移动操作系统iOS、Android以及Windows Phone的浏览器都支持HTML5，这也为HTML5应用开发提供了良好的平台。HTML5技术从诞生之日起就具备跨平台开发的特性，目前国内外已经有很多基于HTML5的跨平台开发工具，开发者并不需要通过编写原生代码来开发跨平台应用。一些HTML5的相关书籍，通过CSS和JavaScript运用工具中所提供的各种丰富的功能模块，可在很短的时间内完成App的开发，并使其具备完美的用户体验。HTML5技术让移动开发变得更简单、更快速、更经济。但是，目前市场上关于HTML5应用开发的书籍并不多，且这些书也不适合直接作为教材。为了便于读者学习HTML5应用开发技术以及相关高校课程的开设，我们结合多年从事HTML5应用开发的经验，编写了本书。

HTML5 App Design and Development

HTML5 App 应用开发教程

黄波 张小华 黄平 王彩◎编著

清华大学出版社

北京

内 容 简 介

越来越多的公司采用 HTML5 来快速开发移动跨平台 App,它支持当前市场流行的移动设备。本书主要介绍了 HTML5 在移动 App 开发中的应用技术、CSS 3 的应用及 JavaScript 的编程知识,并使用大量实例介绍利用 Hbuilder、MUI、HTML5+ 规范开发 App 的流程和实现。

本书将帮助读者快速学习如何利用 HTML5 和 DCloud 的 HTML5 移动开发技术来开发移动 App,也可以作为对 HTML5 App 实践感兴趣的读者和专业开发人员的参考用书。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

HTML5 App 应用开发教程/黄波等编著. —北京:清华大学出版社,2018

ISBN 978-7-302-48199-7

I. ①H… II. ①黄… III. ①超文本标记语言—程序设计—教材 IV. ①TP312.8

中国版本图书馆 CIP 数据核字(2017)第 209659 号

责任编辑:曾 珊

封面设计:李召霞

责任校对:梁 毅

责任印制:刘海龙

出版发行:清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址:北京清华大学学研大厦 A 座

邮 编:100084

社 总 机:010-62770175

邮 购:010-62786544

投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈:010-62772015, zhiliang@tup.tsinghua.edu.cn

课件下载: <http://www.tup.com.cn>, 010-62795954

印 装 者:北京泽宇印刷有限公司

经 销:全国新华书店

开 本:185mm×260mm 印 张:28.5

字 数:696 千字

版 次:2018 年 1 月第 1 版

印 次:2018 年 1 月第 1 次印刷

印 数:1~2000

定 价:79.00 元

产品编号:071608-01

前言

PREFACE

为什么要写这本书

近年来移动互联网的发展十分迅猛，而 HTML5 开发也受到了各世界顶级软件公司的极力推崇和支持，大量的投资以及苹果公司、谷歌公司、微软公司，W3C 的一次次联盟正说明了这点。目前主流移动操作系统 iOS、Android 以及 Windows Phone 的浏览器都支持 HTML5，也更加凸显了 HTML5 技术在未来移动设备端的地位。

HTML5 技术从诞生以来，就具备跨平台开发的特性，目前国内外已经有很多基于 HTML5 的跨平台开发工具，开发者并不需要任何的原生应用编程经验，只需要一些 HTML 的相关知识，懂一些 CSS 和 JavaScript，运用工具中所提供的各种丰富的功能模块，便可在很短时间内完成 App 的开发，并使其具备完美的原生体验。HTML5 技术让移动开发更简单，更适合开发当今流行的移动应用。

截至 2015 年，已经有 80% 的 App 是全部或部分基于 HTML5 技术的。移动互联网行业的快速发展催生了开发热，各大企业对于 HTML5 开发类人才的需求不断增大，HTML5 App 开发人员的缺口巨大，这也激发了广大编程人员学习 HTML5 App 开发以及众多院校开设这方面课程的热情。

虽然 HTML5 技术在开发市场上已经占有了很大比例，相关技术书籍也不少，但是大部分都集中在传统的网页技术上。对于如何使用它来进行相关 App 开发，却鲜有涉及，并且这些书也不适合直接作为教材。为了便于读者学习 HTML5 App 相关开发技术以及相关高校课程的开设，我们结合这两年授课过程的总结，以及与企业联合定制培养 HTML5 App 相关人才的经验，编写了本书。

在技术选型上，本书选择了 DCloud 数字天堂（北京）网络技术有限公司的 HBuilder IDE、HTML5+ Runtime 和 MUI 框架，它们的完美结合做到了接近原生 App 的功能和体验，给开发者提供了许多便利。截至 2017 年 1 月 1 日，HBuilder 的开发者数量已高达 70 万人，成为中国最主流的 HTML5 开发工具。

经过半年多见缝插针式的奋战，本书终于顺利交稿了，我们感到欣慰，同时也为能将自己多年来参与项目开发和指导的经验以及教学上的心得与各位读者分享而感到高兴。

本书第 1、5、7、12、13、14 章由黄波和王彩编写，第 2、4、6、8 章由张小华编写，第 3、9、10、11 章由黄平编写。书中的源代码由王彩整理。

本书适合作为高等院校计算机及相关专业的教材，也可以作为相关培训机构的培训教

材,以及对 HTML5 App 开发技术感兴趣人员的自学用书。

HTML5 App 开发是一个发展迅速的技术,很多方面还在不断完善和变化。由于能力和水平所限,虽然竭尽全力,但仍然难免存在错误和疏漏,希望各位专家、老师和同学提出问题,与编者共同讨论。编者的邮箱为 html5toApp@163.com。

本书特点

- 1. 内容丰富,由浅入深
本书以“看得懂、学得会、做得出”为原则,系统地介绍了 HTML5 App 开发的各种技术和知识,通过每章的内容逐渐引领读者进入 HTML5 App 的开发世界。
书中所讲解的知识基础而实用,并且课程量适中,能让读者在认真学习本课程后基本具备 HTML5 App 的开发能力,成功进入到 App 开发的世界中。
- 2. 结构清晰,讲解到位
本书中配合每个需要讲解的知识点都给出了丰富的插图与完整的实例,使得初学者易于上手。书中所有实例都是实际开发中的例子,结构清晰明了,便于学习。同时书中还给出了很多关于 HTML5 App 开发的实用技巧与心得,具有较高的参考价值。最后一章还给出了一个综合的 App 开发实例“美食汇”的开发讲解。
- 3. 提供书中所有实例源码
为了便于学习,读者可以方便地从清华大学出版社网站(<http://www.tup.com.cn/>)获取本书配套资源包,资源包中包含了书中所有案例的完整源代码,最大限度地帮助读者快速掌握各方面的知识与技术。对于书中所有需要访问的服务器端 API,我们已经部署在 Internet 上,便于读者方便练习,更集中精力掌握 App 开发的前端相关技术。
- 4. 配套的详细课件和习题
为了便于课堂授课,教师可以很方便地从清华大学出版社网站(<http://www.tup.com.cn/>)获取所有章节对应的 PPT 课件。这大大降低了教师备课的难度和时间成本,使得教师可以更好地把精力集中在教学环节,提高授课质量。同时每章最后都配有精心设计的习题,并提供了相应的答案,便于读者复习和教师出题。

学习建议

本书共分为 14 章,讲解的内容按照由简到难的顺序进行安排。其中包括了 HTML5 App 开发的多方面的知识,课内学时建议 64 学时,具体内容及安排如下表所示:

章 名	主要内容	课内学时	课外学时
第 1 章 HTML5 App 应用开发概述	简要介绍 HTML5 的一些新特性,HTML5 App 与原生 App 的比较,开发环境 HBuilder 的使用	2	1
第 2 章 HTML5 页面基础	了解 HTML 语言特性,介绍 HTML5 文档基本格式,以及用于 HTML5 App 开发的一些常用标签	6	3

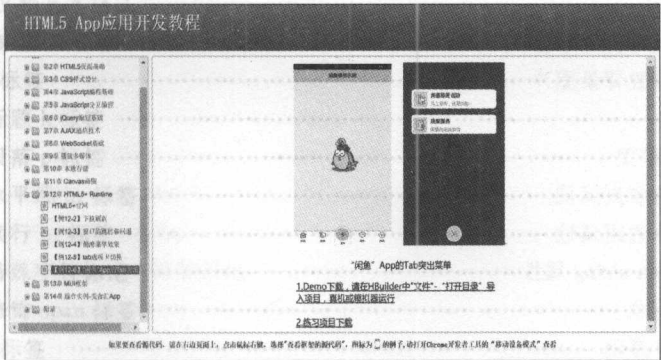
续表

章 名	主 要 内 容	课内学时	课外学时
第 3 章 CSS 样式设计	介绍 CSS 样式规则和使用,在 App 开发中的一些常用 CSS 属性,使用 Chrome 调试 CSS	8	6
第 4 章 JavaScript 编程基础	介绍 JavaScript 的一些基本语法、定义函数、各种内置对象的使用,JSON 数据处理,调试技巧	7	3
第 5 章 JavaScript 交互编程	介绍 JavaScript 的各种 DOM 操作和样式编程	5	3
第 6 章 jQuery 编程基础	介绍 jQuery 的选择器、事件的监听、各种 DOM 交互和功能扩展	5	2
第 7 章 AJAX 通信技术	介绍 AJAX 的技术原理、HTTP 协议,Fiddler 抓包工具、XMLHttpRequest 对象的使用,jQuery 的 AJAX 方法支持,RESTful API 使用	6	3
第 8 章 WebSocket 基础	介绍 WebSocket 的技术原理以及相应的 API	2	2
第 9 章 播放多媒体	介绍 audio 和 video 标签的使用,以及相应的 API	2	2
第 10 章 本地存储	介绍 HTML5 中本地存储技术 localStorage、sessionStorage、Web SQL 和 IndexedDB 的使用	3	3
第 11 章 Canvas 绘图	介绍 Canvas 以及相应绘图 API 的使用	3	2
第 12 章 HTML5+Runtime	介绍 HTML5+ 的模块组成,在页面中使用 HTML5+ API,WebView 模块的一些典型使用	4	4
第 13 章 MUI 框架	介绍 MUI 项目页面的布局、一些内置方法,事件和窗口管理、典型的一些 UI 组件和插件,AJAX 通信调用,在 Chrome 中调试 Android 程序	6	4
第 14 章 综合实例:美食汇 App	给出一个完整的 Android App 开发实例	5	8

注:建议课外学时为 46 学时,便于完成一些实例的练习,任何编程的学习都不能指望在课堂上解决所有的问题,必须在课外进行适时练习。教学或学习过程中可按实际情况对学时和内容进行调整。

本书配套资源包使用说明

下载本书配套资源包的压缩文件,解压后用 Chrome 浏览器打开其中的“index.html”,界面如下图所示,单击左侧树形菜单中各章节每个例子的编号,右侧将显示该例子的效果和说明。



目录

CONTENTS

第 1 章 HTML5 App 应用开发概述	1
1.1 HTML5 介绍	1
1.1.1 终将失败的 Flash	2
1.1.2 Web 移动应用的未来	3
1.2 HTML5 新特性	3
1.3 拥抱 HTML5	6
1.4 HTML5 App 的发展	7
1.5 HTML5 App 与原生 App 的比较	9
1.6 HTML5 App 开发环境搭建	11
1.6.1 开发工具的安装	11
1.6.2 最接近原生 App 体验的 MUI 框架	12
1.6.3 HTML5+应用介绍	14
1.6.4 流应用介绍	15
1.7 开发第一个 HTML5 App	16
1.7.1 练习: HelloWorld 程序	16
1.7.2 打包过程	20
小结	23
习题	23
第 2 章 HTML5 页面基础	25
2.1 HTML 简介	25
2.1.1 标签	26
2.1.2 标签的属性	26
2.1.3 注释标签	27
2.2 HTML5 文档基本格式	27
2.3 布局 div 标签	29
2.4 文本控制标签	30
2.4.1 标题 h 标签	30
2.4.2 段落 p 标签	30
2.4.3 水平线 hr 标签	31
2.4.4 换行 br 标签	31
2.4.5 特殊字符标记	32
2.4.6 修饰 span 标签	32
2.5 图像 img 标签	33

2.6	超链接 a 标签	34
2.7	列表标签	35
2.7.1	无序列表 ul 标签	35
2.7.2	有序列表 ol 标签	36
2.8	语义化标签	37
2.9	页面交互性标签	38
2.9.1	细节展示 details 和 summary 标签	38
2.9.2	进度条 progress 标签	39
2.10	表格标签	39
2.11	表单的应用	42
2.11.1	表单 form 标签	42
2.11.2	各种 input 输入标签	44
2.11.3	input 标签的其他属性	52
2.11.4	其他表单标签	52
2.11.5	实例: 注册表单	54
2.12	移动开发中 meta 标签的应用	56
	小结	57
	习题	57
第 3 章	CSS 样式设计	60
3.1	CSS 简介	60
3.2	CSS 核心基础	61
3.2.1	CSS 样式规则	61
3.2.2	CSS 中的单位和颜色	62
3.2.3	在 HTML 文档中应用 CSS	62
3.3	CSS 选择器	63
3.3.1	基础选择器	63
3.3.2	其他选择器	65
3.4	尺寸属性	65
3.5	文本样式属性	66
3.6	CSS 高级特性	70
3.6.1	继承性	70
3.6.2	CSS 层叠性和优先级	70
3.6.3	Chrome 调试 CSS	71
3.7	背景属性	74
3.7.1	设置背景颜色	74
3.7.2	设置背景图片	75
3.8	边框属性	79
3.9	CSS 盒子模型	80
3.9.1	内填充属性	80
3.9.2	外边距属性	81
3.9.3	box-sizing 属性	82
3.10	浮动和定位	83
3.10.1	浮动	83

3.10.2	定位	85
3.10.3	块元素与行内元素	88
3.11	CSS 动画效果	89
3.11.1	过渡	89
3.11.2	2D 及 3D 变换	90
3.11.3	动画控制	91
3.12	其他一些常用的 CSS 属性	94
3.13	移动设备的适配	95
3.14	实例	97
3.14.1	注册表单样式美化	97
3.14.2	旅游 App 页面	98
	小结	98
	习题	99
第 4 章	JavaScript 编程基础	101
4.1	JavaScript 介绍	101
4.2	使用 JavaScript	102
4.2.1	在页面中插入代码	102
4.2.2	使用 js 文件	103
4.3	JavaScript 的基础语法	104
4.3.1	数据类型	104
4.3.2	变量定义	105
4.3.3	数据类型的转换	106
4.3.4	代码注释	109
4.3.5	运算符	110
4.3.6	常用语句	116
4.4	函数	122
4.4.1	函数定义及调用	122
4.4.2	变量的作用域	123
4.4.3	函数重载	124
4.4.4	函数的返回值	125
4.4.5	匿名函数	126
4.5	JavaScript 在 Chrome 中的调试	126
4.5.1	在控制台输出	126
4.5.2	断点调试	128
4.6	JavaScript 对象基础	130
4.6.1	Object 对象	130
4.6.2	内置对象	131
4.6.3	自定义类或对象	140
4.7	JavaScript 处理 JSON	143
4.7.1	JSON 格式结构简介	143
4.7.2	JSON 序列化与反序列化	143
	小结	145
	习题	145

第 5 章 JavaScript 交互编程	147
5.1 DOM 介绍	147
5.2 使用 DOM	148
5.2.1 document 对象	148
5.2.2 查找节点	150
5.2.3 处理属性	154
5.2.4 读取和设置内容	155
5.2.5 创建和操作节点	158
5.3 DOM 的样式编程	165
5.3.1 className 属性	165
5.3.2 classList 对象	166
5.3.3 style 对象	168
5.4 事件	170
5.4.1 常用的一些事件	170
5.4.2 内联属性监听事件	171
5.4.3 DOM 属性监听事件	172
5.4.4 标准的事件监听函数	172
5.4.5 事件触发过程	175
5.4.6 事件的 Event 对象	176
小结	178
习题	179
第 6 章 jQuery 编程基础	182
6.1 jQuery 介绍	182
6.2 使用 jQuery	183
6.3 使用 \$() 函数	183
6.4 jQuery 的自定义选择器	184
6.4.1 基本过滤器	185
6.4.2 内容过滤器	185
6.4.3 可见性过滤器	185
6.4.4 表单选择器	185
6.5 jQuery 对象与 DOM 对象的转换	186
6.6 jQuery 对事件的处理	187
6.6.1 页面加载后执行	187
6.6.2 jQuery 事件监听	188
6.7 jQuery 遍历方法	193
6.7.1 遍历 HTML 元素对象	193
6.7.2 遍历数组对象	194
6.7.3 遍历 JSON 对象属性	195
6.8 jQuery DOM 交互	195
6.8.1 操作 HTML 属性	196
6.8.2 操作表单元素的值	198
6.8.3 修改节点内容	199
6.8.4 创建和添加 HTML 元素节点	200

6.8.5	删除 HTML 元素节点	202
6.8.6	复制 HTML 元素节点	204
6.8.7	修改样式	205
6.9	jQuery 的扩展	209
6.10	jQuery 插件应用介绍	212
6.11	实例: 记忆翻牌游戏	214
	小结	215
	习题	215
第 7 章	AJAX 通信技术	219
7.1	AJAX 技术介绍	219
7.2	HTTP 协议分析	221
7.2.1	HTTP 协议介绍	221
7.2.2	Fiddler 抓包神器	221
7.2.3	HTTP 请求与响应	225
7.2.4	Fiddler 手机数据抓包	231
7.2.5	Fiddler 模拟 HTTP 请求	232
7.2.6	图片验证码	233
7.3	XMLHttpRequest 对象	235
7.3.1	使用方法	235
7.3.2	读取数据	237
7.3.3	提交数据	239
7.3.4	FormData 对象	241
7.3.5	解析 XML 数据	246
7.4	CORS 跨域问题	248
7.5	RESTful API 介绍	250
7.6	jQuery 中的 AJAX 方法	250
7.7	实例: 送货地址管理	252
	小结	253
	习题	254
第 8 章	WebSocket 基础	257
8.1	WebSocket 的发展历程	257
8.2	HTML5 WebSocket 简介	258
8.3	WebSocket 实现	259
8.4	实例: 聊天室	261
8.4.1	WebSocket 服务器端	261
8.4.2	客户端实现	262
	小结	264
	习题	264
第 9 章	播放多媒体	266
9.1	HTML5 标准中的音视频	266
9.1.1	<audio> 标签	266
9.1.2	<video> 标签	267
9.1.2	图片懒加载插件	390

9.2	<audio>和<video>标签的主要属性	267
9.3	audio 对象和 video 对象的 API	269
9.4	实例: 视频播放器	270
	小结	271
	习题	271
第 10 章	本地存储	273
10.1	HTML5 本地存储技术概述	273
10.2	localStorage 和 sessionStorage	274
10.2.1	检查浏览器的支持	274
10.2.2	相应的 API	275
10.3	Web SQL 数据库	280
10.3.1	创建或打开数据库	280
10.3.2	执行 SQL 语句	280
10.4	IndexedDB 数据库	283
10.4.1	数据库初始化	284
10.4.2	对象存储空间	284
10.4.3	索引	285
10.4.4	事务	285
10.4.5	IndexedDB 的 CRUD 操作	286
10.4.6	游标	286
	小结	288
	习题	288
第 11 章	Canvas 绘图	289
11.1	Canvas 介绍	289
11.2	绘制图形	290
11.2.1	绘制直线	290
11.2.2	绘制贝塞尔曲线	293
11.2.3	绘制填充	294
11.2.4	使用渐变色	295
11.2.5	绘制矩形	297
11.2.6	绘制圆弧	298
11.3	绘制文字	299
11.4	绘制图片	301
11.5	擦除	302
11.6	坐标变换	304
11.7	像素操作	306
11.8	实例: 九宫格手势解锁	307
	小结	308
	习题	308
第 12 章	HTML5+Runtime	310
12.1	HTML5+Runtime 介绍	310
12.2	HTML5+ 的 Demo 示例	311

12.3	HTML5+API 的使用	312
12.4	HTML5+API 的各模块	313
12.5	Webview 模块	314
12.5.1	Webview 的方法	314
12.5.2	WebviewObject	321
12.5.3	常见的一些 UI 效果	326
12.6	Native.js 介绍	340
	小结	340
	习题	340
第 13 章	MUI 框架	342
13.1	MUI 介绍	342
13.2	MUI 的示例	343
13.3	使用 MUI	344
13.4	MUI 页面设计的一些特殊使用	346
13.5	mui 对象的内置方法和对象	350
13.6	事件管理	352
13.7	窗口管理	355
13.8	各种 UI 组件	362
13.8.1	按钮	362
13.8.2	数字角标	365
13.8.3	数字输入框	365
13.8.4	列表	366
13.8.5	折叠面板	369
13.8.6	卡片视图	370
13.8.7	轮播组件	371
13.8.8	图片轮播组件	371
13.8.9	复选框和单选框组件	374
13.8.10	开关组件	374
13.8.11	滑块组件	375
13.8.12	字体图标组件	376
13.8.13	表单组件	376
13.8.14	进度条组件	378
13.8.15	弹出菜单组件	380
13.8.16	遮罩层组件	381
13.8.17	操作表组件	381
13.8.18	对话框组件	382
13.8.19	scroll 区域滚动	385
13.9	下拉刷新和上拉加载	386
13.9.1	下拉刷新	387
13.9.2	上拉加载	388
13.10	MUI 的插件	389
13.10.1	延迟加载插件	389
13.10.2	图片预览插件	390

13.10.3	日期和时间选择器插件	391
13.10.4	单页面刷新插件	392
13.11	MUI的AJAX封装	394
13.12	Chrome调试Android应用	395
	小结	397
	习题	398
第14章 综合实例：美食汇App		399
14.1	项目介绍	399
14.1.1	API全局变量	400
14.1.2	API介绍	400
14.2	字体图标的制作	401
14.3	manifest.json文件的配置	403
14.3.1	应用信息配置	403
14.3.2	图标配置	404
14.3.3	启动图片配置	405
14.4	向导	406
14.5	首页	409
14.5.1	使用子页面构建首页	409
14.5.2	美食列表数据的请求和刷新	410
14.5.3	滑动手势的处理	412
14.5.4	过滤条件的制作	412
14.5.5	MUI插件的使用	413
14.5.6	窗口数据的传递	414
14.5.7	扫码的实现	414
14.5.8	城市定位和选项卡切换	415
14.6	美食详情	416
14.6.1	拨打电话	417
14.6.2	百度地图定位显示	417
14.6.3	评论中的图片预览	419
14.6.4	分享	419
14.6.5	收藏	423
14.7	抽奖	424
14.7.1	授权打开窗口	424
14.7.2	界面处理	425
14.7.3	摇一摇	426
14.7.4	自定义窗口	427
14.7.5	跨页面调用方法	427
14.8	注册和登录	428
14.9	我的订单	430
14.9.1	artTemplate生成列表	430
14.9.2	评论	432
14.10	版本更新	435
14.11	发布Android程序	437
	小结	439
部分习题参考答案		440
参考文献		442

HTML5 App 应用 开发概述

学习目标

- 了解 HTML5 标准以及它的新特性。
- 了解 HTML5 App 的发展及其与原生 App 的比较。
- 掌握开发环境的搭建。
- 掌握 HBuilder 移动 App 项目的开发,动手实现 HelloWorld 项目,并实现云打包。

HTML5 标准的制定、硬件的提升、浏览器引擎的不断升级,为 HTML5 App 的发展带来了契机。HTML5 自问世以来,其良好的跨平台兼容性让它受到了前所未有的关注,并成为移动平台开发技术最重要的一员,本章将针对 HTML5 作简单介绍,并对 HTML5 App 开发的优秀工具 HBuilder 作详细的讲解。

1.1 HTML5 介绍

HTML(Hyper Text Markup Language,超文本标记语言)是互联网发展的基石,目前几乎所有的网站都是基于 HTML 进行开发的。HTML5 是 HTML 的最新标准,经过近 8 年的艰辛努力,于 2014 年 10 月由 W3C(万维网联盟)发布为正式推荐标准,接下来,这门互联网编程语言也将走上更加规范化的道路。2016 年 9 月底,W3C 发布了 HTML 5.1 标准,并宣布着手编写 HTML 5.2 的语言标准规范。

十年前,Web 只是用来展示一个文档,而现在 Web 则成为了应用程序的一个运行平台。互联网的不断发展对网站的功能提出了很多更高的要求,但由于 HTML 没有及时地跟进这些需求,很多厂商或组织在 HTML 上各自建立了自己的标准,如 Flash、SilverLight、JavaFx 等。因为商业上的原因,这些标准往往很难被广泛接受,于是造成了各种解决方案互相竞争的局面。

当前,移动互联网时代 Web 开发主要面临两种困境:

(1) 有些功能必须借助合适的插件才能实现,例如目前流行的页面上音频和视频的播放大多借助于 Flash 插件。

(2) PC 端和移动端应用的多次开发,必须为微软、苹果、安卓等系统设计不同的方案,这意味着一个创业团队或公司必须招纳不同的技术人才,因此大大提高了运营成本。

HTML5 标准的制定过程正值移动互联网崛起,标准组织成员中的 Apple、Google、

Opera 本身便有着对移动互联网的独立思考和见解,并最终影响 HTML5 的实际成果。设计之初,HTML5 便拥有桌面互联网、移动互联网兼容并蓄的想法,不仅统一了开发方式、网络内容,还做到了访问方式的体验统一化。

1.1.1 终将失败的 Flash

2011 年 11 月,Adobe 正式宣布停止为移动设备的浏览器开发 Flash Player,转而全面发展 HTML5 技术,并表示“HTML5 是各种移动平台浏览器中最佳的内容制作和发布解决方案”,正式宣告了 Flash 退出移动端开发的舞台。所以,目前大多数视频网站采用的是双重技术,如果在 iOS 上欣赏网站视频,都会自动切换到 HTML5 播放模式上,PC 上则采用 Flash 插件。

Flash 插件是指安装于浏览器的插件(Adobe FlashPlayer Plugin),使浏览器得以播放 swf 格式的文件,目前常见的网页广告、一些特效、音视频的播放都是借助于 Flash 插件实现的。在过去的 Web 时代,Flash 确实如日中天,成为桌面浏览器的霸主;但是到了移动互联网时代,苹果公司推出的 iPhone、iPad 却拒绝使用 Flash,其原因有以下几点。

1. Flash 的不开放

Flash 是一个完全封闭的系统,Adobe 公司拥有 100% 的技术专利,Adobe 想通过 Flash Player 授权来收费,每台移动设备嵌入 Flash Player,预收 1 美元,Adobe 还希望开发者用 Flash 来编写 iOS 移动设备上的软件。苹果的 iPhone、iPad 和 iOS 生态体系也是封闭的,内容的营收体系均建立在 iOS 的 App Store 模式之下。两者都想完全控制内容生态,这无疑会产生巨大的利益冲突。

苹果公司选择了 HTML5、CSS 和 JavaScript,它们全都是开放标准。苹果公司的所有移动设备都与生俱来地对这些开放标准有着良好的支持,网页开发者利用 HTML5 就能做出高级的图像、字体、动画以及过渡效果,而不必依赖 Flash 插件。HTML5 完全开放,并受 W3C 标准委员会控制,苹果公司是该委员会的成员之一。

2. 安全性和性能

Flash 的安全性漏洞较多,黑客经常利用这些漏洞。2009 年,赛门铁克公司曾经指出,Flash 是最不安全的系统之一。有资料表明,苹果电脑死机的罪魁祸首就是 Flash,这对于看重安全性的 iOS 系统是完全不能容忍的,2014 年苹果公司就曾在 Mac OS X 系统上远程屏蔽 Adobe Flash Player 的所有版本文件。几年过去了,Flash 的安全性漏洞问题依然很多。

为了在播放视频时保持良好的电池续航力,移动设备必须用硬件来对视频进行解码,苹果公司的工程团队当时没有向 Flash 工程团队开放 Flash 调用苹果显卡 GPU 加速的能力,结果可想而知,Flash 在苹果设备上全部都通过 CPU 方式来计算和渲染,造成了 Flash 的性能低下,电池续航能力也大幅下降。

3. 触屏支持

Flash 是为个人电脑和鼠标设计的,并不适合触屏。许多 Flash 网站都用到了光标悬停:当用户把光标移动到某个点时,弹出菜单或其他元素。苹果公司革命性的多点触控界面不用鼠标,也没有光标悬停的概念。如果要支持触屏设备,大部分 Flash 网站都要重写。而如果开发者要重写 Flash 网站,为什么不用更新的 HTML5 技术呢?就算 iPhone、iPod

和 iPad 支持 Flash, 还是不能解决大多数 Flash 网页需要重写以便支持触屏设备的问题。

1.1.2 Web 移动应用的未来

移动设备的广泛使用, 使许多传统开发者很无奈。一个企业真的既需要一个 Web 站点, 又需要针对每个主流的移动平台都有一个移动应用程序? HTML5 的可移植性以及所有的移动平台上的良好表现, 让开发者看到了希望: 有许多开发工具可以让他们利用现有的技能, 不管是 Web 移动应用三剑客——HTML5、CSS 和 JavaScript, 还是像 Java 或 Object-C 那样的流行编程语言, 都可以用来为 iOS、Android、Windows Phone 创建应用, HTML5 将不只是下一代 Web 开发标准, 基于 HTML5、CSS 和 JavaScript 的移动应用程序才是未来的趋势。HTML5 移动应用开发并不是单指 HTML5 标准, 而是指 HTML5+CSS3+JavaScript 三项技术(如图 1-1 所示)的集合——HTML5 负责内容, CSS3 负责外观, JavaScript 负责行为。

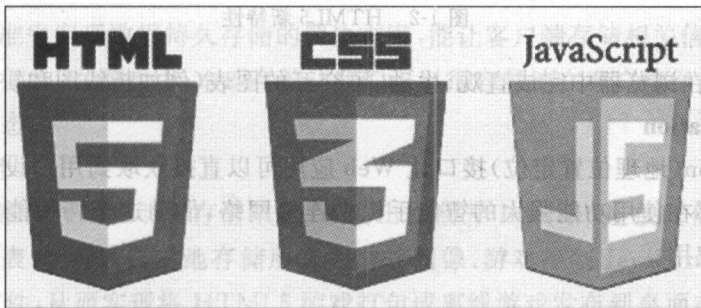


图 1-1 Web 移动应用三剑客

1.2 HTML5 新特性

HTML5 是最近十年来 Web 开发标准的巨大飞跃, 它引入很多新的特性, 带来了不一样的全新体验, 如图 1-2 所示。

1. 各种语义化标签

根据内容的结构化(内容语义化), 选择合适的标签(代码语义化)便于开发者阅读和写出更优雅的代码, 同时让浏览器的爬虫机器人很好地解析页面, 便于实现更好的 SEO (Search Engine Optimization, 搜索引擎优化)。

2. 不需要插件的音视频支持

旧的 Web 标准中没有对音视频的支持, 基本都是借助于其他插件(如 Flash)来实现的。HTML5 标准的出现解决了这个令人头疼的问题, 目前无论是在桌面, 还是移动平台上的浏览器, 都可以良好地支持音视频的播放。

3. Canvas

Canvas 是 HTML5 标准中的一个在底层提供绘制图形以及操作位图功能的元素, 相当于浏览器上的画布, 基于 JavaScript 用它来绘制图形、图标, 以及其他任何视觉性图像, 也可以创建图片特效和动画, 它在 HTML5 游戏中也扮演了一个很重要的角色。另外, 依靠

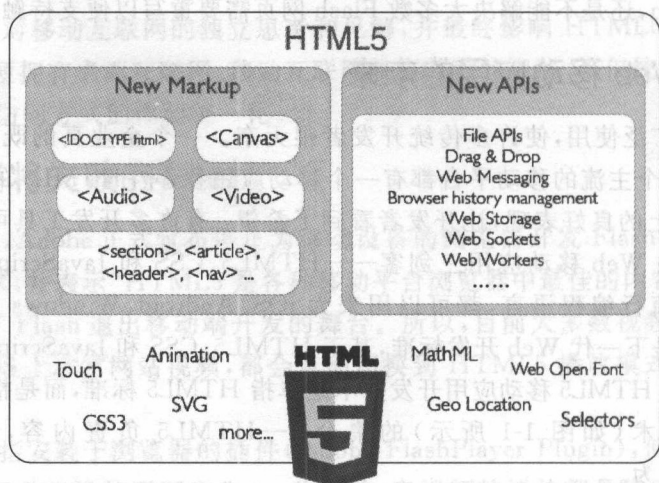


图 1-2 HTML5 新特性

Canvas,还可以在浏览器中完成直观、生动、可交互的图表(例如折线图和饼形图等)。

4. Geo Location

GeoLocation(地理位置定位)接口让 Web 应用可以直接获取到用户设备的经纬度。越来越多的用户都在使用功能强大的智能手机来连接网络,借助这个特性能够开发出很多基于位置信息的应用。

5. WebGL

WebGL 扩展了 Canvas 元素,为 Web 浏览器提供了一套 3D 图形 API,这套 API 基于 OpenGL ES 2.0 标准。基于 WebGL,可利用底层的图形硬件实现加速渲染,在浏览器中展现各种 3D 模型,实现各种超炫的效果。

图 1-3 是 zego body(<https://zygotbody.com/zb>)在 Chrome 浏览器中的截图,它演示了如何使用 WebGL 处理一个 3D 的人体模型,人体模型可以旋转、缩放、加上或去掉肌肉。

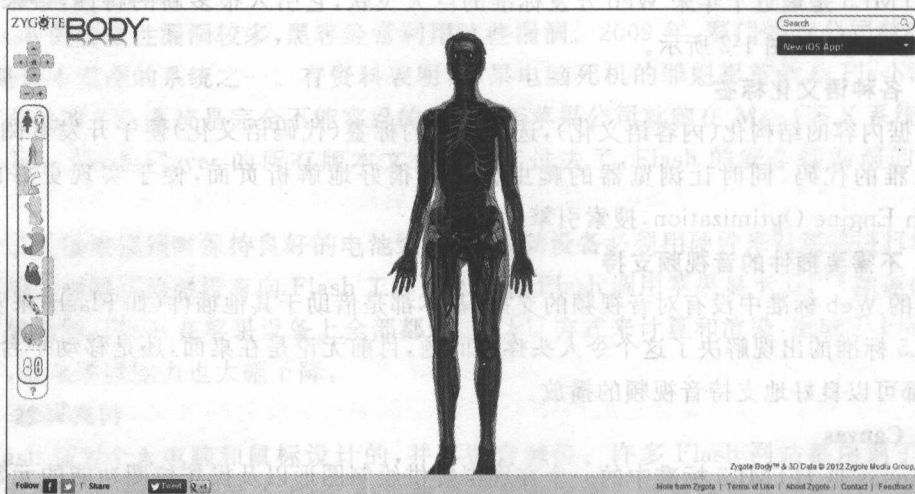


图 1-3 3D 人体模型

图 1-4 是一款基于 WebGL 实现的很逼真的水波动画,画面上是一个大水池,水池底部是一颗大石头,点击水面时即可泛起水波,加上模拟光的照射,也可以拖动石头让其在池底滚动,也可以拖动画面实现多视角观看(<http://madebyevan.com/webgl-water/>)。

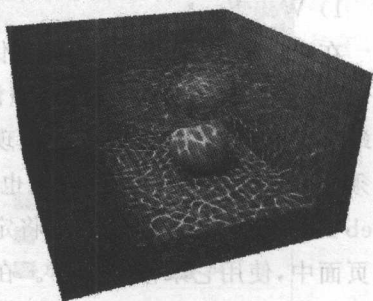


图 1-4 逼真的水波动画

6. WebSocket

WebSocket 是 HTML5 标准的一部分,Web 页面可使用它来实现持久连接到 Socket 服务器上,该接口提供了浏览器与服务器之间的事件驱动型连接,实现浏览器与服务器之间的双向通信。当服务端有数据更新时,服务器就可以直接将数据推送到客户端。

7. 本地存储

HTML5 标准中实现数据持久存储的解决方案,能让客户端存储相关信息,并在需要的时候进行读取。例如,在游戏中,本地存储让我们可以很容易地在客户端保存游戏进度和最佳成绩等游戏状态。

8. 离线应用

HTML5 标准中可以声明缓存清单,用于列出在断开 Internet 连接时依然能访问页面所需要的文件列表,就可以在本地存储所有的游戏图像、游戏控制 JavaScript 文件、CSS 样式和 HTML5 文件,从而实现将 HTML5 游戏打包成离线游戏发布到桌面或移动设备上的功能。

9. Web Worker

Web Worker 是 HTML5 提供的一个 JavaScript 多线程解决方案,可以将一些大计算量的代码交由 Web Worker 运行而不阻塞用户界面。

10. File API

在早期的浏览器技术中,处理少量字符串是 JavaScript 最擅长的,但文件处理,尤其是二进制文件处理,一直是空白。HTML5 的 File API 允许浏览器访问本地文件系统,借助 File API,浏览器将具备更强大的文件处理能力。

11. Drag&Drop

早期,各前端工程师为了实现浏览器中元素的拖放功能可说是煞费苦心,在 HTML5 中,拖放是标准的一部分,任何元素都能够拖放。

12. Web Messaging

Web 开发过程中常常会碰到一个问题:跨域通信(Cross-Domain communication)。HTML5 Web 消息机制就是 JavaScript 开放给浏览器的 API,能够让 HTML5 页面之间传递消息,甚至这些页面可以在不同的域名下。

13. CSS3 新功能

CSS 是表现层,它定义了如何展现 HTML5 页面。在 HTML5 时代,CSS3 的一些新特性让页面得以使用简单的方式实现一些更复杂的特效。

1) Web Font

在以前的 Web 开发中,为了保证页面效果,往往需要实现一些特殊的字体效果,但没有办法确定用户客户端是否支持,只能采用图片实现。遇到需要修改时,必须重作图片,采用这种方式也降低了页面的效率。Web Font 可以帮助开发人员将定义的 Web 字体嵌入到页面中,使用它来修饰文本。在用户进行访问时,会自动将相应的字体下载到客户端上。

2) 动画

图 1-5 展示了一个使用 CSS3 控制的行走动画效果,这种效果在以前必须借助于 Flash 插件实现,实现过程也较为复杂,而 CSS3 简单高效地实现了这个动画效果。

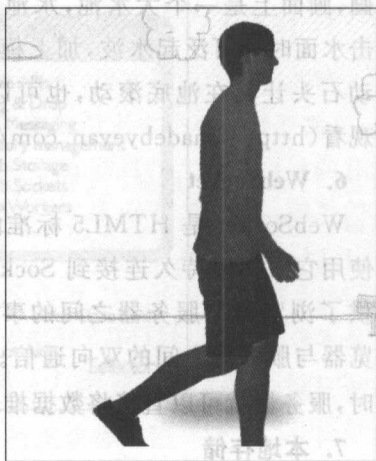


图 1-5 行走动画

1.3 拥抱 HTML5

当前的浏览器能多大程度支持 HTML 5 新标准呢? 在线网站 HTML5test 可以根据浏览器支持 HTML5 规范的程度来对浏览器打出相应的分数(网址为: <http://html5test.com/>), 满分为 550 分, 你的浏览器也可以和其他浏览器进行对比, 如图 1-6 所示。

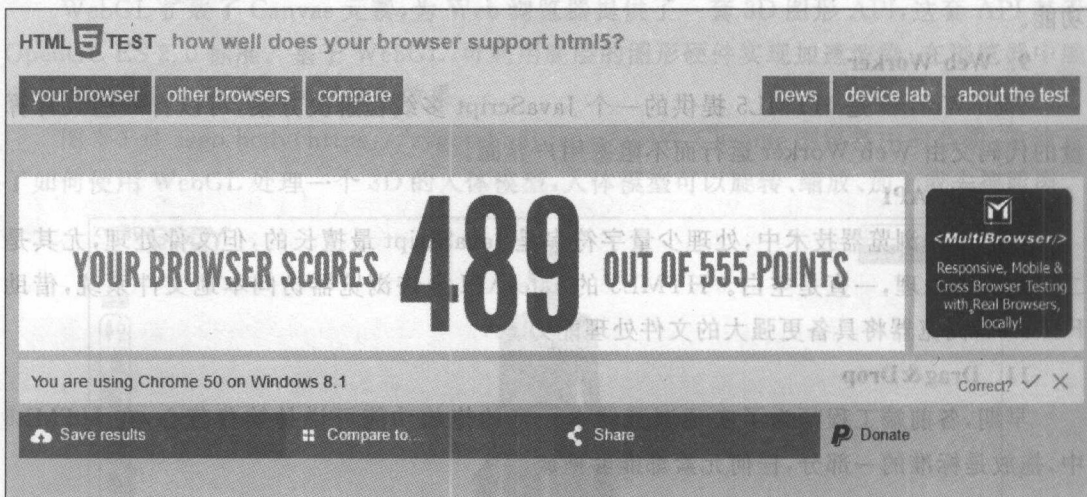


图 1-6 HTML5 跑分

从图 1-7 和图 1-8 可以看出,目前无论是 PC 端还是移动端的浏览器,得分越来越高,这意味着它们对 HTML5 的支持程度也越来越高,这项得分将成为未来衡量浏览器优劣的一个重要指标。特别是微软公司的 IE,在这上面一直得分不高,微软果断抛弃了 IE,重新设计了 Edge 引擎,极大地提高了浏览器性能和 HTML5 兼容性。

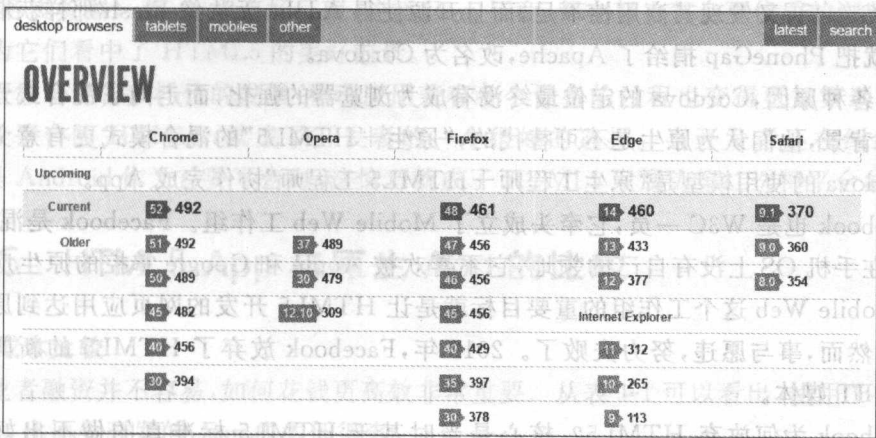


图 1-7 PC 端各大浏览器的得分

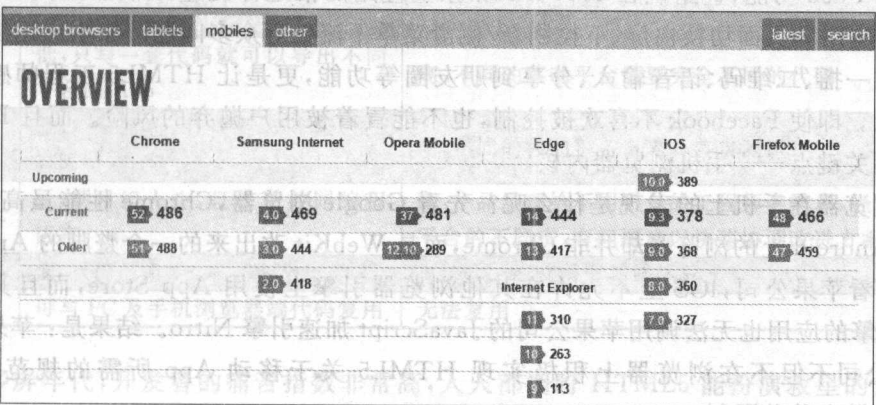


图 1-8 移动端各大浏览器的得分

1.4 HTML5 App 的发展

1. 过去

在移动互联网时代,HTML5 的跨平台优势在移动互联网时代被进一步凸显。HTML5 是唯一一个适用于 PC、Mac、iPhone、iPad、Android、Windows Phone 等主流平台的跨平台语言。Java 和 Flash 都曾梦想这个位置,但梦断于 iOS。人们纷纷开始研究基于 HTML5 开发跨平台手机应用。很多人当时认为,原生应用只是过渡,就像当年从 C/S 结构转变为 B/S 结构一样。而且开发者学习 Objective-C 和 Java 很困难,既然会网页开发,为何不试试 HTML5 呢?!但是移动互联网初期的更新太快了,手机 OS 在不停地扩展硬件,如陀螺仪、距离感应器、气压计,每年手机 OS 都有大版本更新。W3C 作为一个数百家会员单位共同决策的组织,从标准草案的提出到达成一致是非常复杂的过程,跟不上移动互联网初期的快速更新。所以一直未能拿出成熟标准。

PhoneGap 的出现给开发者打开了一扇窗。很多人期待 PhoneGap 不停扩展 API,来补充浏览器的不足。Adobe 看到 PhoneGap 仿佛看到了重振江湖地位的希望,但在 Adobe 收

购 PhoneGap 后,又发现其商用性不足,而且开源使得 Adobe 无法像 Flash 那样获取商业利益,于是就把 PhoneGap 捐给了 Apache,改名为 Cordova。

因为各种原因,Cordova 的定位最终没有成为浏览器的强化,而走向了混合式开发。基于当时的背景,他们认为原生是不可替代的,“原生+HTML5”的混合模式更有意义。所以现在 Cordova 的使用模型是“原生工程师+HTML5 工程师”协作完成 App。

Facebook 也是 W3C 一员,它牵头成立了 Mobile Web 工作组。Facebook 是混 Web 圈的,并且在手机 OS 上没有自己的领地,它不喜欢被 Apple 和 Google 掌控的原生应用生态系统。Mobile Web 这个工作组的重要目标就是让 HTML5 开发的网页应用达到原生应用的体验。然而,事与愿违,努力失败了。2012 年,Facebook 放弃了 HTML5 的新闻充斥了全世界的 IT 媒体。

Facebook 为何放弃 HTML5? 核心是当时基于 HTML5 标准真的做不出好的移动 App。对比 Twitter 等竞争对手的原生 App,Facebook 的 HTML5 版本实在无法让用户满意。例如 Push 功能,直到现在 HTML5 和原生应用的推送体验差距依然巨大,更不用说 HTML5 应用的页面切换白屏、下拉刷新/侧滑菜单不流畅等众多问题。原生应用工程师轻松实现摇一摇、二维码、语音输入、分享到朋友圈等功能,更是让 HTML5 工程师感觉自己站错了队。即使 Facebook 不喜欢被控制,也不能冒着被用户抛弃的风险。而且 Facebook 没有掌握关键点——手机浏览器内核。

而浏览器在手机上的表现是什么呢? 先看 Google 浏览器,Chrome 性能虽高,但早期版本的 Android 上的浏览器却并非 Chrome,而是 WebKit 改出来的一个蹩脚的 Android 浏览器;再看苹果公司,iOS 上不允许在其他浏览器引擎上使用 App Store,而且其他使用 Safari 引擎的应用也无法调用苹果公司的 JavaScript 加速引擎 Nitro。结果是:苹果公司和 Google 公司不但不在浏览器上积极实现 HTML5 关于移动 App 所需的规范,反而对 HTML5 做出种种限制。

不管是当时硬件能力不足,还是手机 OS 厂商的故意限制,结果是:在移动互联网的初期,是原生应用生态系统的天下。

2. 现在

不管是站在用户的角度,还是站在开发者的角度,HTML5 App 的发展都是移动互联网发展的潮流。2011 年,iPhone 4s 的 CPU 是 A5,现在的 iPhone 6 则变成了 A8,速度提升了 7.5 倍,解决了很多 HTML5 的性能问题。Google 在 2013 年年底发布的 Android 4.4,内置的 WebView 不再是 Android WebKit 浏览器,而是 Chromium,使其性能大幅提升。从最新的 Android 5.0 开始,WebView 可以通过 Google Play Store 实时更新,和 Chrome 的升级保持一致,用户不刷机就可以享受到最新的浏览器引擎;再看 Apple 方面,2012 年 iPhone 5 发布后,HTML5 在 iOS 上的表现已令人满意,Safari 独家的 JavaScript 加速引擎 Nitro 不再那么重要,不过在 iOS 8 发布后,苹果公司还是很识趣地取消了第三程序调用 Nitro 的限制,现在任意浏览器或应用调用 iOS 的 UIWebView 都可以利用 Nitro 加速,这样在前端使用 JS 进行大型运算也成为可能。两大手机操作系统霸主和浏览器巨头的态度发生了变化,使得 HTML5 在手机上的发展不再受限,而且这个变化不可逆,只能继续向前,这种变化势必会产生深远的影响。

拥有大流量入口的互联网巨头,都希望内嵌更优秀的增强引擎。腾讯推出了 X5 浏览

器引擎,各浏览器厂商、应用市场厂商,甚至 Rom 厂商,都在努力整合更优质的浏览器引擎,正是因为它们看中了 HTML5 的美好前景。

随着移动设备多样化的发展,移动应用兼容各个设备的过程也变得更加繁琐。而能够横跨多个平台的 HTML5,就成了开发者的一个很好的选择。当开发者还在纠结是选择 iOS 还是 Android 作为首要平台时,这恰巧就成了 HTML5 发展为第二选择平台的契机。

1.5 HTML5 App 与原生 App 的比较

1. 成本比较

创业者融资并不容易,如何花钱更高效非常重要。从表 1-1 可以看出,使用 HTML5 开发的 App 比原生开发的 App 成本低得多。

表 1-1 成本比较

	HTML5 App	原生 App
开发成本	低,只写一套代码就可以导出不同平台的 App	高,不同的移动平台需要完全不同的代码
测试成本	低,一个工程,代码行数有限,bug 有限	高,不同的移动平各自需要各自测试
沟通成本	低,项目组人少,且使用共同编程语言	高,产品经理给不同平台的工程师讲需求,经常出现的平台的不同步,有问题难以快速协调改进方案
招聘成本	低,入门门槛低,人才基数大	高,人才少,又需要招聘不同平台
复用	可与 PC 及手机浏览器端代码复用	无法复用

在多屏年代,开发者的痛苦指数非常高,人人都期盼 HTML5 能扮演救星的角色。多套代码、不同技术工种、业务逻辑同步,这是折磨人的过程。有点类似个人电脑早期世界,那个时候的电脑都有各自的操作系统和编程语言,开发者疲于做不同版本。跨平台技术在早期大多因为性能问题而夭折,但中后期硬件能力增强后又会占据主流,因为跨平台确实是刚需。

2. 难度比较

从难度来看,由于 HTML5 都是统一采用 JavaScript,而 Web 程序员是编程世界中数量最多的一个群体,基本都可以无障碍或只需少量培训就可以进行 HTML5 App 开发,从表 1-2 可以看出,原生 App 程序员的培养成本要高得多。

表 1-2 难度比较

	HTML5 App	原生 App
学习难度	低,脚本语言,描述式标签和样式	高,编译语言,需要理解很多与业务逻辑无关的底层技术
开源资源	多,有非常多的开源库,不用“重复发明轮子”	相对少
招聘难度	低,人才基数大	高,人少,优秀的人更少。抢不过 BAT

3. 迭代速度比较

移动互联网时代是一个“快鱼吃慢鱼”的时代,谁对用户的需求满足得更快,谁的试错成

本更低,谁就拥有巨大的优势。互联网产品大多免费,且有网络效应,后入者抢夺用户的难度非常大。从表 1-3 可以看出,使用原生开发,从招聘、开发、上线各个环节的效率都很慢。

表 1-3 迭代速度比较

	HTML5 App	原生 App
上线	快,先拿到用户的产品,让对手更难拿到用户	慢,从招聘、开发、上线各个环节的效率都慢一倍以上,而且参与的人越多,沟通效率越低下
debug	快,可以直接 debug 任意界面	慢,只能从主入口进入,在子界面调试(swift 有所改进)
试错	快,灰度发布和 A/B 测试更灵活	慢,调整麻烦,实验成本高
招聘成本	低,入门门槛低,人才基数大	高,人才少,又需要招聘不同平台
更新	快,更快地满足用户需求,更快地解决用户不满	慢,出了问题即使很快改好,也要等应用商店受理更新

很多开发者有这样的体会,一个原生应用在 App Store 上线后,如果发现有严重的 bug,开发者就需要连夜加班修复,然后静待 2 周或更长时间的 Apple 审核,等新应用通过审核后上线,用户基本上已经流失了。但是,HTML5 没有这些问题,可以实时更新,快速响应。

4. 市场推广比较

由于超级 App 的巨大流量能轻易成为 HTML5 应用的入口,从表 1-4 可以看出,未来它们会形成更大的市场效应。传统的应用商店、甚至线下预装,这些流量不足和效率偏低的发行模式将被挤出市场主流。即使是超级 App 的大流量应用商店,如果转型得当,也将以发行 HTML5 应用为主。

表 1-4 市场推广比较

	HTML5 App	原生 App
用户获取应用的途径	多,超级 App(如微信朋友圈)、搜索引擎、应用市场、浏览器,到处都是 HTML5 的流量入口	少,主要是应用市场
入口流量	大,上述各种入口累计流量非常大,尤其是社交类超级 App	相对小,主要是应用市场的流量,偶尔有广告互推荐的流量
导流效率	高,页游和端游打同样的广告,广告变用户的转化率,页游远远高于端游	低,用户从看到应用的广告到装好该应用的门槛和时间都很长,导致折损率高

未来的市场中,原生的广告和统计 SDK 提供商也会面临尴尬,Google、百度等基于网页的广告和统计服务会取得更大的优势。开发者不再需要打包 SDK,只需引入一个 Script 即可。

5. 结论

对于开发者来说,HTML5 App 的开发效率更高,开发快,更新快,更能适应“快鱼吃慢鱼”的移动互联网市场变化;成本更低,从技术到管理,各项成本大大降低,便于创业,也降低了失败的机率和风险;对于产品的推广也更容易,导流入口多,效率高,流量大。对于用户而言,App 应用的获取,更快速方便,更省流量;App 的使用更省电量,更省空间。

HTML5 技术正在席卷 App 行业,如何高效率地开发出轻架构、高性能的 HTML5

App 已是抢占潮流红利的关键。

1.6 HTML5 App 开发环境搭建

在开发 HTML5 App 之前,首先需要在系统中搭建开发环境。HBuilder 是 DCloud(数字天堂)网络技术有限公司推出的一款支持 HTML5 的 Web 开发 IDE。HBuilder 主体是由 Java 编写,它基于 Eclipse,兼容了 Eclipse 的插件。这款 IDE 具有以下特点:

- (1) 最快的 HTML5 开发工具, 编码比其他工具快 5 倍;
- (2) 最全的语法库和浏览器兼容库;
- (3) 令人震撼的语法提示, 无死角提示, 除了语法, 还能提示 ID、Class、图片、链接、字体等;
- (4) 支持手机 App 开发, 真机或模拟器调试, 可以边改边看和 Run in device;
- (5) 支持云打包或本地打包, 一套程序可以打包成 Android 或 iOS 安装包;
- (6) 环保的视觉主题;
- (7) 丰富的插件;
- (8) 多种语言支持: php、jsp、ruby、python、nodejs 等 Web 语言, less、coffee 等编译型语言均支持。

目前 HBuilder 已经生成了十几万个移动 App(不包括离线打包数据)。

1.6.1 开发工具的安装

首先到 DCloud 公司的官网(<http://www.dcloud.io/>)下载开发工具 HBuilder,目前的版本是 7.4.0,界面如图 1-9 所示,分别提供了 Windows 版本和 MacOSX 版本。这个开发工具是绿色软件,不需要安装,而且完全免费。本书主要以 Windows 版本为例进行讲解。下载解压后,进入相应的 HBuilder 目录,将 HBuilder.exe 发送到桌面快捷方式。软件启动后,第一次会显示用户登录界面,如果没有账号,就先注册一个,软件会自动检测是否有新版

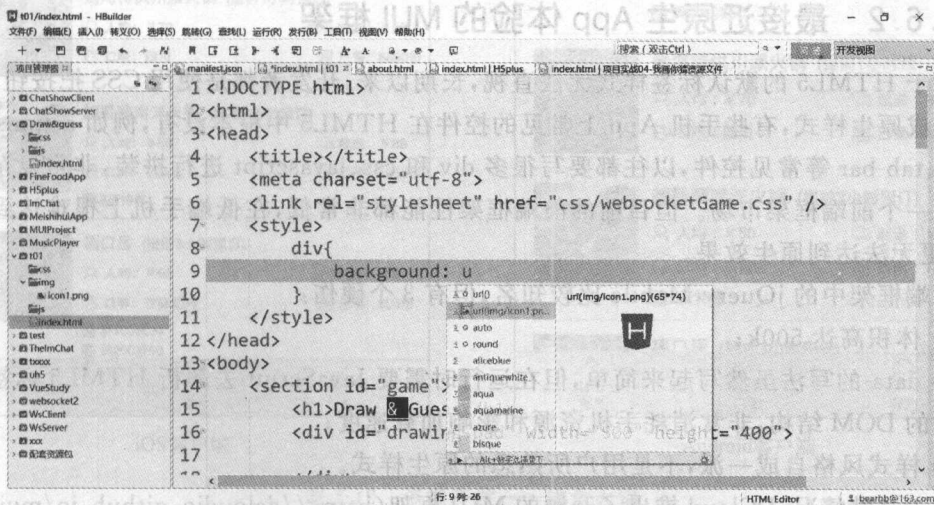


图 1-9 HBuilder 界面

本,并提示升级。

在百度中输入“Chrome 浏览器”,下载谷歌浏览器最新官方版本,采用默认安装。Chrome 浏览器的界面如图 1-10 所示。对于 HTML5 的开发,Chrome 浏览器的 DevTools 工具有强大的功能和友好的用户体验,不仅能快速方便调试 JavaScript、检查 HTML 页面 DOM 结构、实时同步更新元素 CSS 样式,还能跟踪分析页面资源加载性能等问题。对于移动平台的开发者来说,从 Android 4.4 开始,也可以通过 Chrome 的 DevTools 工具连接设备对应用进行调试。



图 1-10 Chrome 浏览器界面

1.6.2 最接近原生 App 体验的 MUI 框架

由于 HTML5 的默认标签样式无法直视,长期以来,开发者都是使用 CSS 把按钮、输入框修饰成原生样式,有些手机 App 上常见的控件在 HTML5 中根本没有,例如 actionsheet、switch、tab bar 等常见控件,以往都要写很多 div 和 css、javascript 进行拼装,非常复杂。这引发了一个前端框架市场。但目前的前端框架性能都非常低,在低端手机上很难达到商用要求,更无法达到原生效果。

前端框架中的 jQuery Mobile 比较知名,但有 3 个硬伤:

- ① 体积高达 500k;
- ② data-的写法虽然写起来简单,但在运行时需要 JavaScript 去解析 HTML5 标签并替换为新的 DOM 结构,非常消耗手机资源和影响加载速度;
- ③ 样式风格自成一派,不是用户所熟悉的原生样式。

基于这种情况,DCloud 推出了开源的 MUI 框架(<http://dcloudio.github.io/mui/>),它是目前最高性能和最接近原生体验的手机端框架。它的 3 个特点与 jQuery Mobile 正好

对应:

① 体积小;

② 直接使用 class 编写,性能远高于 data-方式,又通过代码块的编写方式降低了开发者编码的复杂度,在 HBuilder 输入 m,会拉出一排控件 mList、mButton 等,按 Enter 键,就会自动产生 div 和 class;

③ 风格样式是最接近原生样式的,如图 1-11 所示。

基于 MUI(<http://www.dcloud.io/mui.html>)有一套 HTML5 工程,通过前端构建工具(如 grunt)条件编译,可同时发行到 iOS App Store、安卓各大应用商店、普通手机浏览器、微信 App 和流应用。图 1-12 展示了使用 MUI 开发的一个精彩案例——“挑食火锅”,它使用了一套 HTML 模板,使用 grunt 作为条件编译,发布出各平台中的 App,用户体验完全一致,这是原生 App 很难做到的,特别是在微信中,原生 App 根本无能为力。

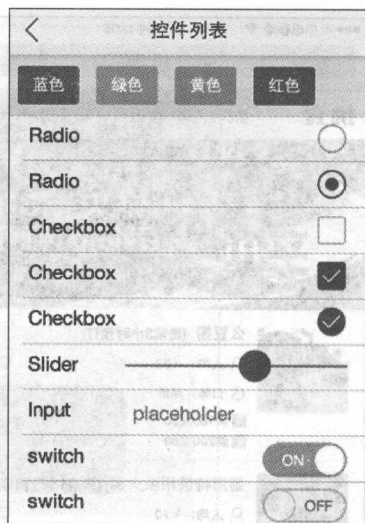


图 1-11 强大的 MUI 前端框架



iOS客户端



Android客户端

图 1-12 多端齐发——挑食火锅



图 1-12 (续)

1.6.3 HTML5+应用介绍

通过 HTML5 开发移动 App 时,会发现 HTML5 很多能力不具备。为弥补 HTML5 能力的不足,在 W3C 的指导下成立了 HTML5 中国产业联盟,推出 HTML5+ 规范。HTML5+ 规范是一个开放规范,它允许第三方浏览器厂商或其他手机 runtime 制造商实现, <http://www.html5plus.org/doc/h5p.html>,图 1-13 是 HTML5+ 的应用架构。

HTML5+ 扩展了 JavaScript API,使得 JavaScript 可以调用各种浏览器无法实现或实现不佳的系统能力,设备能力(如摄像头、陀螺仪、文件系统等),业务能力(如上传下载、二维码、地图、支付、语音输入、消息推送等),实现与原生 App 同样强大的功能和性能。

HBuilder 内置 HTML5+ App 开发环境,提供一套完整的移动应用开发解决方案。内置 HTML5+ API 语法提示,提高开发效率;集成真机运行环境,方便开发后即时在真机上查看运行效果;集成应用云端打包系统,不用部署 Xcode 和 Android SDK 就可以打包应用。开发者只需要使用 HTML5、JavaScript、CSS 技术就可以快速开发跨平台的移动应用。

HTML5+ 规范中的 Native.js 是另一项创新技术。手机 OS 的原生 API 有四十多万,大量的 API 无法被 HTML5 使用。Native.js 把几十万原生 API 封装成了 JS 对象,通过 JS 是目前最高性能和最接近原生体验的手机端技术。它的 3 个特点与 jQuery Mobile 正

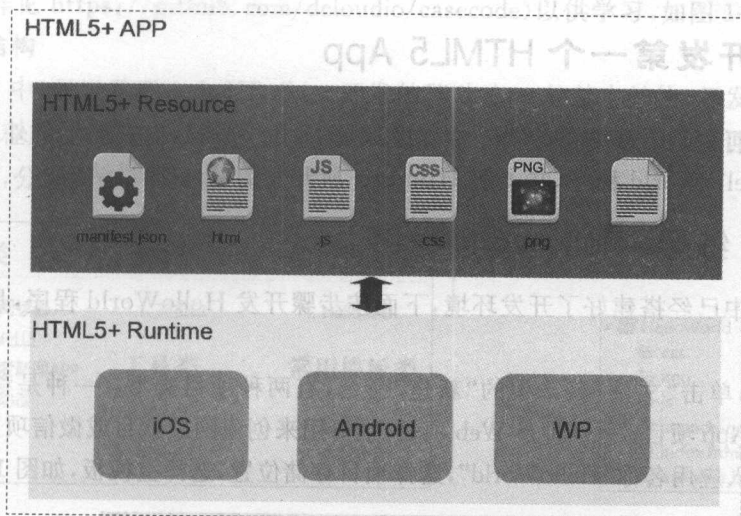


图 1-13 HTML5+应用架构

可以直接调用 iOS 和 Android 的原生 API。这项技术支持 iOS 5.0 及以上和 Android 2.3 及以上版本。

1.6.4 流应用介绍

通过 HTML5 做跨平台 App，以往都是创业公司或外包公司在做。一线的互联网公司不太在意研发成本，即便 HTML5+ 已经可以达到原生的功能和体验，也很少使用 HTML5 做 App。流应用不是一个开发产品，而是一个发行产品，它从另一个角度给一线互联网公司进入 HTML5 阵营提供重要动力。

流应用结合了原生 App 和 Web App 的优势，又消除了各自的缺点。基于 HTML5+ 的程序可以达到原生的效果，同时 DCloud 的专利流式发行技术实现让 App 类似流媒体——可以边使用边下载，免除过往从下载到安装的多步操作，实现 5s 内完成 App 的安装和启动。以往安卓应用市场下载原生 App 时，从单击开始，经历下载—启动安装包—确认权限—解压安装—启动应用等多个流程，根据主流应用市场的数据，从下载到 App 真正激活启动，将折损 50% 的用户。100 次下载中只有 50 个真正启动了 App，而流应用把激活率从 50% 提升至 95%。

目前拥有 6 亿手机用户的 360 手机助手已经集成了 DCloud 的流应用引擎，可以发行流应用。可以用 360 手机助手直观感受一下，打开它之后，选择“分类”菜单，可以发现其中已经有很多流应用的发布，选中一个，单击“秒开”按钮，会发现 App 已经直接打开，退出后会发现桌面上已经生成了应用的图标，方便下次再打开，时间不超过 5s。一些优秀 App 的流应用版本的图标如图 1-14 所示依次为：大众点评外卖、京东秒杀、唯品会、蘑菇街、携程酒店、途牛机票等，它们都推出了相应的流应用版本。



图 1-14 优秀 App 的流应用版本

1.7 开发第一个 HTML5 App

几乎用任何一门语言编写的第一个程序都是 HelloWorld,本书也不例外。本节就教大家如何编写 HelloWorld 程序,并了解 HBuilder 所开发的 App 项目结构。

1.7.1 练习: HelloWorld 程序

在 1.6 节中已经搭建好了开发环境,下面按步骤开发 HelloWorld 程序,具体步骤如下。

1. 创建

打开 IDE,单击“文件”菜单中的“新建”命令,有两种项目类型:一种是移动 App,用来创建 HTML5 App 项目;另一种是 Web 项目,主要用来创建网站项目或微信项目。选择“创建移动 App”,输入应用名称“HelloWorld”,选择项目存储位置,选择空模板,如图 1-15 所示。

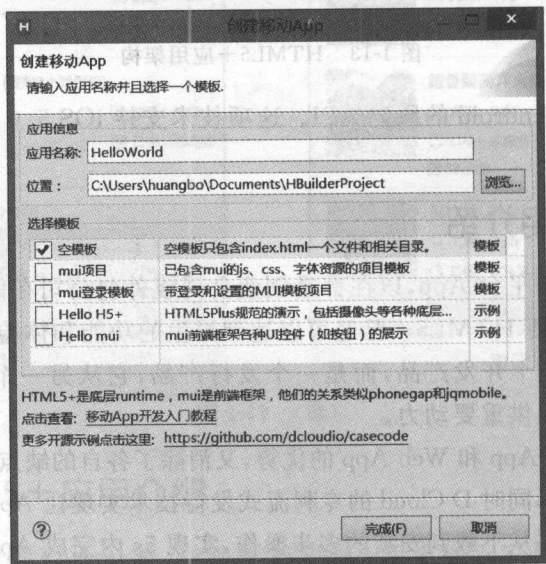


图 1-15 创建移动 App



从图 1-15 可以看出,有不少模板类型,接下来针对这些模板类型进行简单介绍。

- 空模板: 只包含了个一个简单的 index.html 和相关目录,针对完全使用自己的资源开发 App 这种情况,例如以前有相应的手机网站,想直接转成手机 App 应用。
- mui 项目: 使用 DCloud 的 MUI 框架开发手机 App,项目自动包含了 MUI 所用到的 CSS、js、字体资源文件。
- mui 登录模板: 一个示范性的 MUI 项目,完成了常见的注册、登录功能以及界面。
- Hello H5+: HTML5+ 规范的示范性项目,包含了所有的 HTML5+ 规范中各 API 的调用。
- Hello mui: MUI 示范项目,主要是展示各种 UI 控件的使用。

在对话框底部,开发人员可以单击“移动 App 开发入门教程”迅速熟悉 HBuilder 开发 App 的过程(详见 <http://ask.dcloud.net.cn/article/89>),DCloud 公司也提供了一些优秀

的开源项目(详见 <https://github.com/dclaudio/cascode>)以供学习,如图 1-16 所示。

2. 程序结构

在图 1-17 中,可以看到一个 HBuilder 开发的移动 App 的基本结构,开发者可以在此结构上开发应用程序,图标显示为 ,如果是 Web 项目,则图标显示为 ,项目由多个文件以及文件夹组成,分别用于不同的功能:

资讯类	电商类	能力展示类
1. 新闻阅读器 2. JFinal社区 3. 开发者新闻APP 4. 红旅动漫	1. 灰狐E3 工具类 1. 滴石 2. 卉卉打卡	1. Hello MUI 常用模板类 1. 登录模板 2. 酒店预定

图 1-16 一些开源项目



图 1-17 HelloWorld 程序结构

- index.html: HTML5 App 的页面文件,可以根据需要自行添加其他页面或修改。
- css 目录: HTML5 App 所有的.css 文件都放置在这个目录中。
- img 目录: App 所用到的图片都放置在这个目录中。
- js 目录: HTML5 App 所有的.js 文件都放置在这个目录中。
- unpackage 目录: 这个目录比较特别,图标和其他目录也不一样,当文件放入时,打包和真机运行都不会同步那个目录,例如一些参考或项目说明文档。
- manifest.json 文件: HTML5 App 的配置文件,包括应用信息、图标、启动图片、SDK、模块权限等。双击该文件,会打开应用配置界面,如图 1-18 所示。

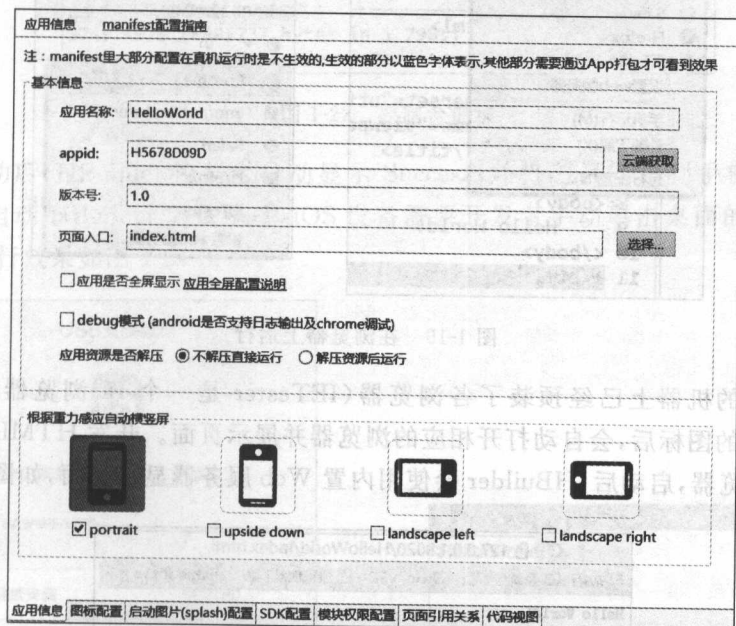


图 1-18 manifest.json 配置界面

3. 代码示范

打开 index.html,修改代码,具体如下所示:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="initial-scale=1.0,maximum-scale=1.0,
    user-scalable=no" />
  <title></title>
</head>
<body>
  Hello World!
</body>
</html>
```

4. 运行

1) 在浏览器上运行

HTML5 App 和 Web 的页面在本质上是相同的,也是由 HTML5 页面、CSS、JavaScript 组合而成,不同的是 HTML5 App 是将所有资源打包成安装包,而 Web 页面是通过远程请求得到页面,两者都可以在浏览器中预览,特别是在设计界面时,经常需要先借助浏览器。可以通过菜单栏中的“运行”菜单启动,也可以通过工具栏启动,如图 1-19 所示。

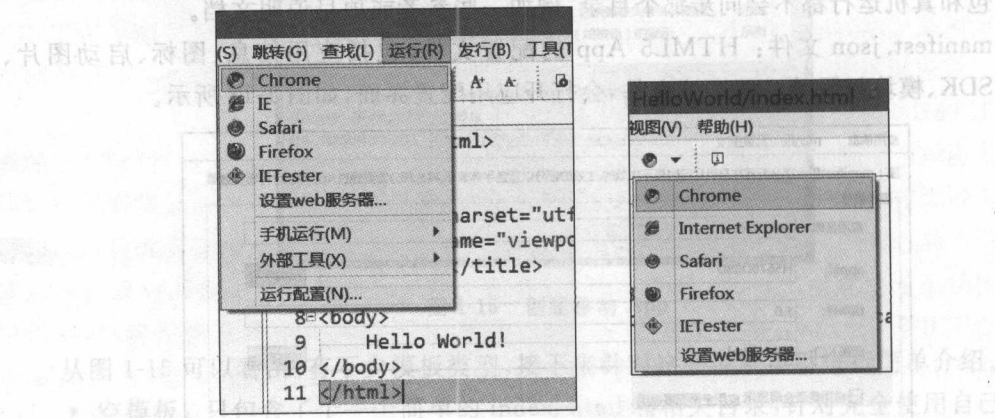


图 1-19 在浏览器上运行

如果开发的机器上已经预装了各浏览器(IETester 是一个 IE 浏览器多版本测试工具),单击相应的图标后,会自动打开相应的浏览器并展示页面。开发 HTML5 App,主要使用 Chrome 浏览器,启动后,HBuilder 会使用内置 Web 服务器显示页面,如图 1-20 所示。

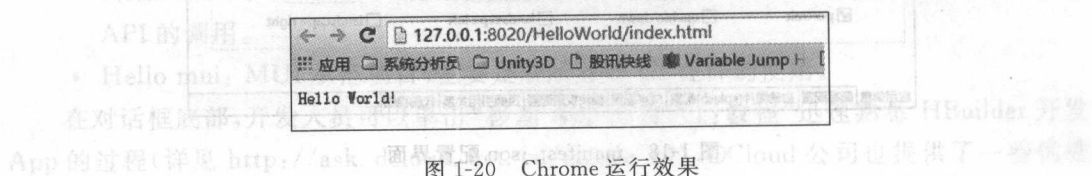


图 1-20 Chrome 运行效果

2) 启动真机运行

真机运行有 3 个特点:

- 真实。虽然 PC 端 HBuilder 的“边改边看”也可以看到大致的页面,但真实的布局效果以及手机上的特殊能力调用仍有差异,还是必须在真机测试。
- 边改边看。在 HBuilder 更改页面并保存后,可立即同步在真机上看到保存后的显示效果。比开发原生应用还方便。
- 检查错误和 log。手机运行 HTML 等文件时,如果发生错误以及打印的 console.log,都可以在真机运行时从手机端反馈回到 HBuilder 的控制台,在控制台直接查看。

将 iOS 或 Android 设备连接到计算机,这时 HBuilder 会自动检测连接到计算机上的设备,通过菜单栏中的“运行”菜单启动,也可以通过工具栏启动,如图 1-21 所示。

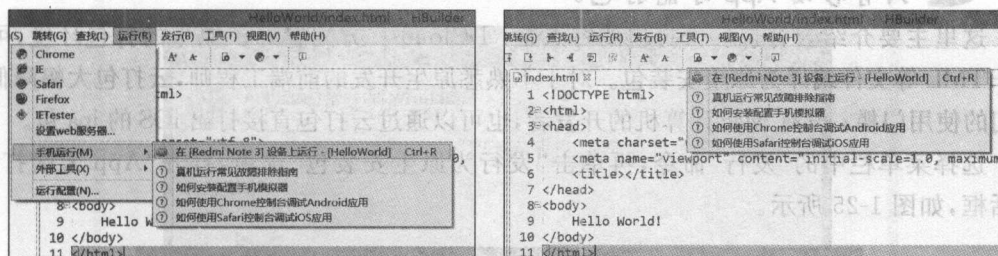


图 1-21 真机运行

启动后,HBuilder 控制台会提示需要在真机上安装一个调试基座程序 android_base.apk,同时手机出现“USB 安装提示”信息,选择“继续安装”,如图 1-22 和图 1-23 所示。

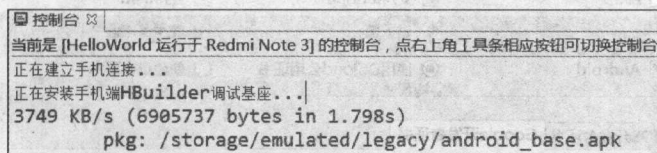


图 1-22 控制台提示

安装成功后,HBuilder 控制台自动显示 Success,并将项目同步到手机端,在 Android 设备会自动启动 HBuilder 调试基座,iOS 设备需要开发者手动单击桌面的 HBuilder 调试 App,手机运行效果如图 1-24 所示。

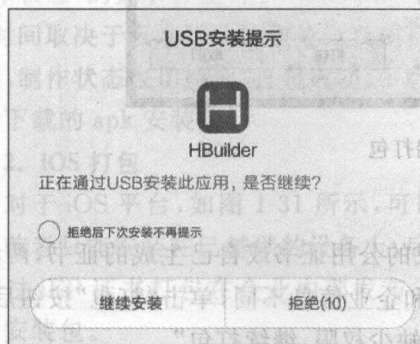


图 1-23 手机提示

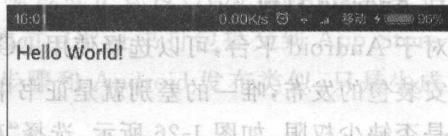


图 1-24 会Android手机运行效果图

试着在 HBuilder 中修改 Hello World 文字,按 Ctrl+S 键运行保存后,会发现手机端的显示也会自动修改,按下手机上的返回键,App 会自动退出。



只有移动 App 才能真机运行。

1.7.2 打包过程

第 1.7.1 节的 HelloWorld 程序只是在手机上测试运行了,如果需要发布到 App Store 和安卓应用市场,还需要将程序打包。DCloud 提供云打包和本地打包两种方式,云打包并不会向开发者收取任何有关打包的费用,开发者也可以使用本地打包。



只有移动 App 才能打包。

这里主要介绍云打包。云打包的特点是: DCloud 官方配置好了原生的打包环境,可以把 HTML 等文件编译为原生安装包。对于不熟悉原生开发的前端工程师,云打包大幅降低了他们的使用门槛;没有 Mac 计算机的开发者,也可以通过云打包直接打出 iOS 的 ipa 包。

选择菜单栏中的“发行”命令,再单击“发行为原生安装包”选项,打开“App 云端打包”对话框,如图 1-25 所示。

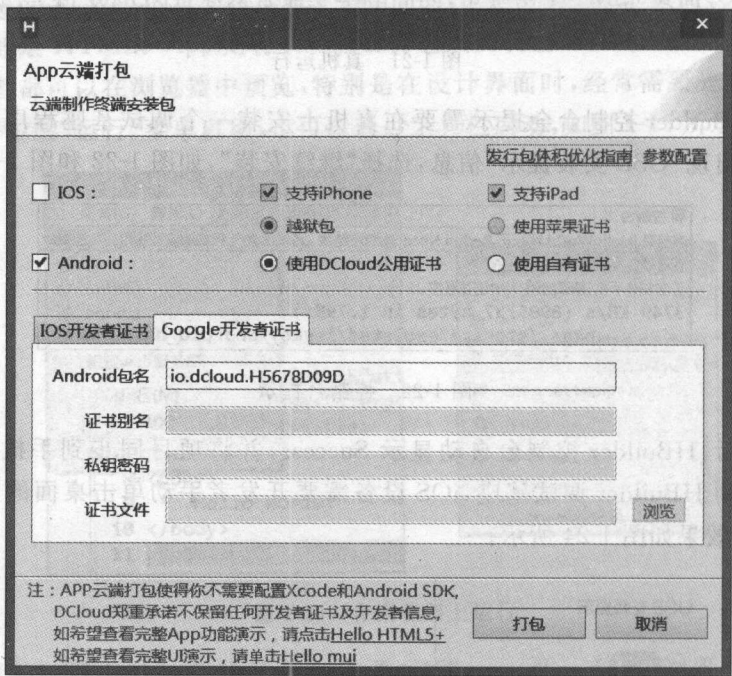


图 1-25 App 云端打包

1. Android 打包

对于 Android 平台,可以选择使用 DCloud 生成的公用证书或自己生成的证书,两者不影响安装包的发布,唯一的差别就是证书中开发者和企业信息不同,单击“打包”按钮后,将提示是否缺少权限,如图 1-26 所示,选择“确认没有缺少权限,继续打包”。

在图 1-27 中, HBuilder 会自动检测云端打包状态,打包过程也可以由控件在后台运行,

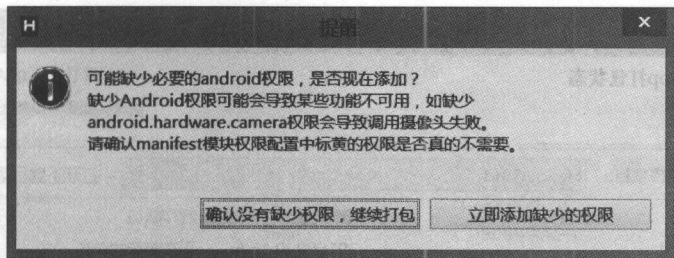


图 1-26 缺少权限提示

也可以单击“详细信息”按钮进行查看,当成功提交到云端后,会提示 App 云端打包成功,开发者可选择是否查看打包状态,可以单击“确定”按钮进行查看,如图 1-28 所示。

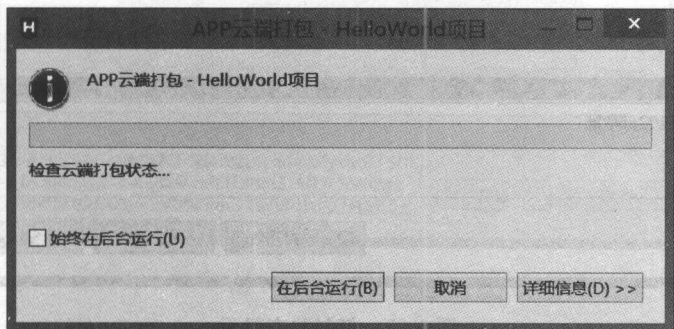


图 1-27 状态检测

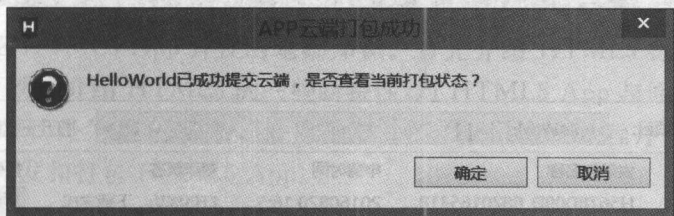


图 1-28 查看打包状态提示

图 1-29 中显示了正在打包的项目 HelloWorld,包括版本号、申请时间、制作状态,其中“制作状态”的显示会在“正在准备”和“正在制作安装包”之间切换。打包过程需要一段时间,时间取决于云端排队的数量以及项目的大小。在图 1-30 中,制作成功后,安装包名称会显示,制作状态也切换为“打包成功,下载完成”,可以单击“打开下载目录”选项,得到打包后自动下载的 apk 安装包。

2. iOS 打包

对于 iOS 平台,如图 1-31 所示,可以选择越狱包或正式包(App Store 专用或企业证书),前者只能安装在已越狱的设备上,后者则可通过 iDP 证书打包提交到 App Store 发布或通过 iEP 证书打包在企业内部发布。其他的步骤和 Android 发布类似,只是生成的是 .ipa 安装包。

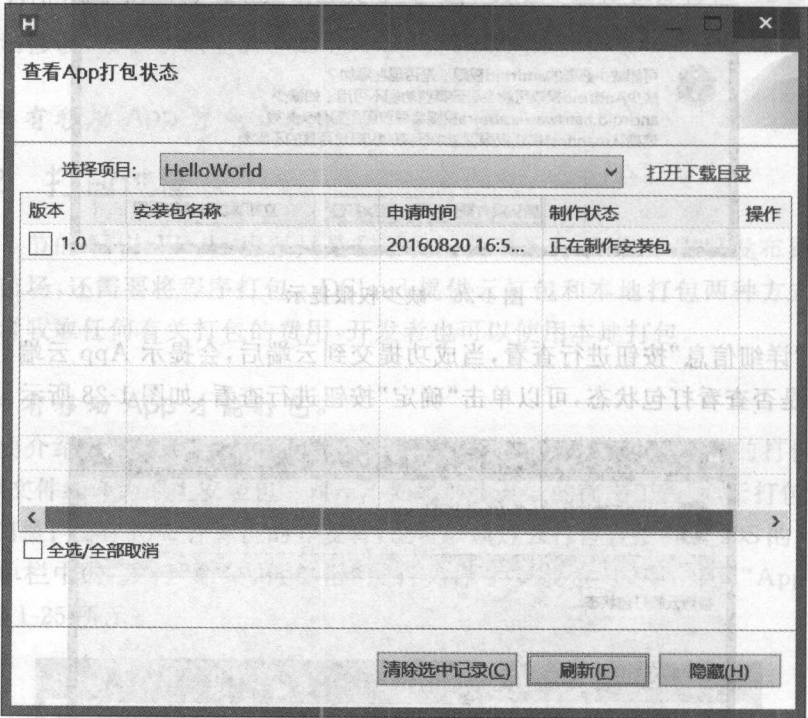


图 1-29 打包状态显示

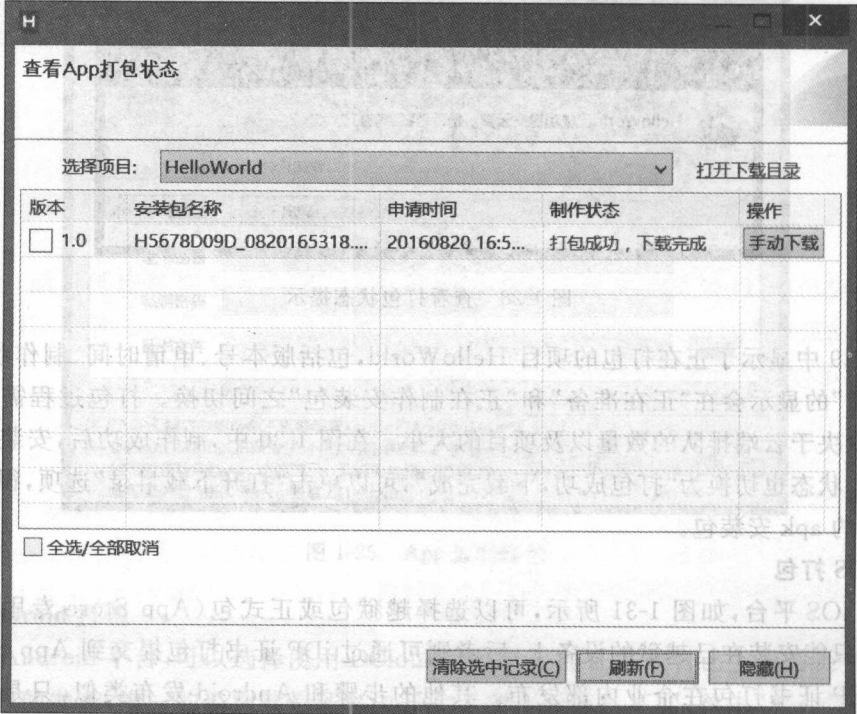


图 1-30 打包成功

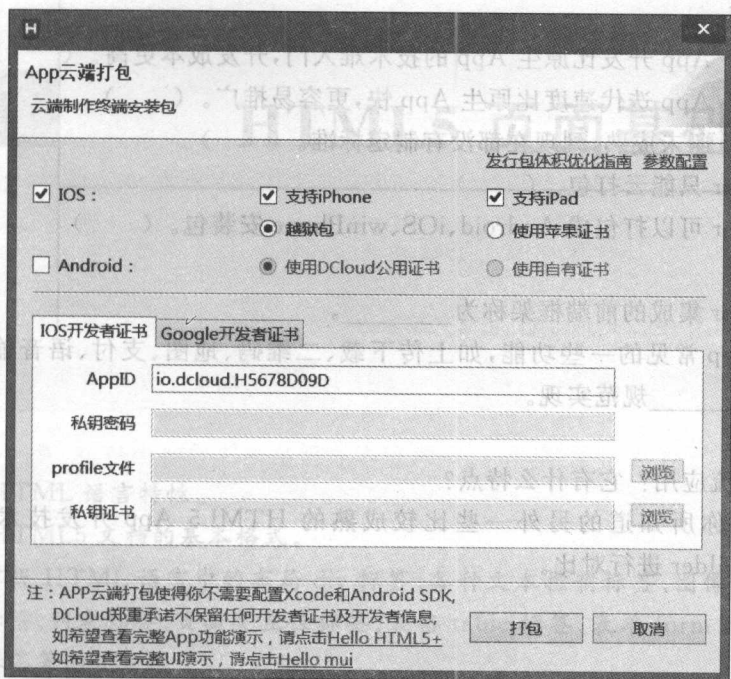


图 1-31 iOS 云打包

小结

本章主要介绍 HTML5 App 开发的基础知识。首先介绍 HTML5 标准,解释了 Flash 必然失败的原因;然后介绍 HTML5 的一些新特性,对 HTML5 App 与原生 App 进行了比较,强调 HTML5 App 是发展的必然;最后通过一个 HelloWorld 程序来讲解如何使用 HBuilder 工具来开发和打包 HTML5 App。

习题

一、选择题

- 下列()选项不属于 HTML5 标准的新特性。
A. canvas B. Geo Location C. 支付 D. WebGL
- 未来 Web 发展方向的三剑客包括()。
A. HTML5 B. CSS3 C. Web Socket D. JavaScript
- HTML5 标准中实现多线程的技术是()。
A. Web Socket B. WebGL C. Weg Messaging D. Web Worker
- 使用 HBuilder 开发 HTML5 App,下面()文件是用来进行配置的()。
A. index.html B. manifest.json C. css D. unpackage
- 使用 HBuilder 开发 HTML5 App,可以生成()的安装包。
A. apk B. ipa C. xap D. rar

二、判断题

- 1. HTML5 App 开发比原生 App 的技术难入门,开发成本更高。()
- 2. HTML5 App 迭代速度比原生 App 快,更容易推广。()
- 3. HTML5 还未成熟,到现在都没有制定标准。()
- 4. HBuilder 只能云打包。()
- 5. HBuilder 可以打包成 Android、iOS、winPhone 安装包。()

三、填空题

- 1. HBuilder 集成的前端框架称为_____。
- 2. 对于 App 常见的一些功能,如上传下载、二维码、地图、支付、语音输入、消息推送等,可以基于_____规范实现。

四、简答题

- 1. 什么是流应用? 它有什么特点?
- 2. 请给出你所知道的另外一些比较成熟的 HTML5 App 开发技术,并把它们与 DCloud 的 HBuilder 进行对比。

第 2 章

CHAPTER 2

HTML5 页面基础

学习目标

- 了解 HTML 语言特性。
- 掌握 HTML5 文档的基本格式。
- 熟练掌握 HTML 语言中的布局 div 标签、各种文本控制标签、图像 img 标签、超链接 a 标签、列表标签、页面交互性标签、表格 table 标签、表单 form 标签、各种 input 输入标签等的使用。
- 掌握 meta 标签在移动应用开发中的应用。

HTML5 App 的界面是由 HTML 页面完成的,而页面上的一切内容都是由 HTML 语言控制实现的。本章会针对 HTML5 的文档基本格式,各种 HTML 标签进行详细讲解,以帮助读者掌握 HTML 的使用,迅速完成 App 页面内容的构建。

2.1 HTML 简介

一个 HTML5 App 是由多个窗口组成,每个窗口就是一张 HTML5 页面,HTML5 页面的扩展名是 .html 或 .htm(例如在第 1 章中完成的“HelloWorld 程序”的主页面是 index.html),页面上所展示的内容就是由 HTML(Hyper Text Markup Language,超文本标记语言)所构建的。

HTML 语言诞生于 1993 年,是一种设计网页的标准语言。所谓超文本,是指页面内可以包含图片、链接,甚至音乐、程序等非文字元素。正是有了它,计算机世界才显得更丰富多彩,而我们也才能畅游在 Internet 世界。HTML 通过标签符号来标记要显示的网页中的各个部分。网页文件本身是一种文本文件,通过在文本文件中添加各种标签符号,可以告诉浏览器如何显示其中的内容(如:文字如何处理、画面如何安排、图片如何显示等)。浏览器按顺序阅读网页文件,然后根据标签符号解释和显示其标记的内容,对书写出错的标签并不指出其错误,也不停止其解释执行过程,我们只能通过显示效果来分析出错原因和出错部位。但需要注意的是,不同的浏览器对于同一标签可能会有不完全相同的解释,因而可能会有不同的显示效果,特别是 PC 上和移动设备上的不同显示最为明显。

HTML 语言是标签标记式语言,所以非常简单,易于掌握,最重要的是它与平台无关,无论是在 PC、Mac、Android,还是 iOS 平台上,只要使用浏览器或浏览器内核都可以浏览和

运行,在 HTML5 时代,它甚至在智能电视上也有一席之地。

2.1.1 标签

标签是由一对尖括号包裹的单词构成的,它们可以被浏览器解释,从而决定页面的结构和显示的效果。标签中的单词书写是不分大小写的,但推荐使用小写字母。标签通常是成对出现的,分为开始标签和结束标签,在它们中间的是**标签体**,其语法格式如下:

```
<标签名>标签体</标签名>
```

有些标签功能是比较简单的,只使用一个标签即可,这种标签叫**自结束标签**,语法格式稍有区别:

```
<标签名/>
```

标签是可以嵌套的,但是注意一定不要出现交叉嵌套,例如下面这种情况:

```
<标签名 1>
```

```
<标签名 2>
```

```
</标签名 1>
```

```
</标签名 2>
```

错误的嵌套

正确的嵌套

```
<标签名 1>
```

```
<标签名 2>
```

```
</标签名 2>
```

```
</标签名 1>
```



如果标签的拼写错误,浏览器不会报错,所以对于标签名的记忆还是必要的。

2.1.2 标签的属性

标签的属性通常出现在标签名的开始标签中,或者是自结束标签中,通常是以键值对的形式存在,属性名字母都是小写,属性值必须以双引号或单引号包裹后,用等号赋值给属性名,语法格式如下:

```
<标签名 属性名="属性值">标签体</标签名>
```

或

```
<标签名 属性名="属性值"/>
```



属性的使用比较特殊,可以使用自定义的属性存储值,对界面不会造成任何影响。

另外,如果标签的属性名和属性值完全一样时,可以直接书写属性名,属性值可以省略,以下面的标签属性写法为例:


```
<input type="text" readonly="readonly"/>
//属性和属性值完全相同,简化为
<input type="text" readonly/>
```

2.1.3 注释标签

在 HTML 中还有一种特殊的标签——注释标签。如果需要对 HTML 代码添加一些便于阅读和理解但又不需要在页面中显示的注释文字,就需要使用注释标签。语法格式为:

```
<!-- 注释语句 -->
```

需要注意的是,注释内容不会显示在浏览器窗口中,但是作为 HTML 页面源代码的一部分,它会自动下载到客户端,在查看源代码时可以看到。



HTML 源代码是可以直接查看的: 用 Chrome 浏览网页时,单击鼠标右键,选择“查看网页源代码”。

2.2 HTML5 文档基本格式

学习任何一门语言,都需要首先掌握它的基本格式,就像写信需要符合书信格式要求一样。HTML5 的页面也不例外,同样需要遵从一定的规范。下面讨论 HTML5 文档的基本格式的特点。

打开 HBuilder,选中第 1 章的 HelloWorld 项目,单击鼠标右键,选择“新建”命令中的“HTML 文件”选项,弹出“创建文件向导”窗口,如图 2-1 所示,文件名输入 test.html,选择 html5 模板后,单击“完成”按钮,就可以得到一张空白的 HTML5 页面(HTML5 App 中的新窗口也是这样添加的),它的代码如下所示:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
  </body>
</html>
```

这些自带的 HTML 源代码构成了 HTML5 页面文档的基本格式,下面对它们作具体的介绍。

1. DTD 声明

代码的第一行<!DOCTYPE html>是 HTML 文档的 DTD(Document Type Definition,

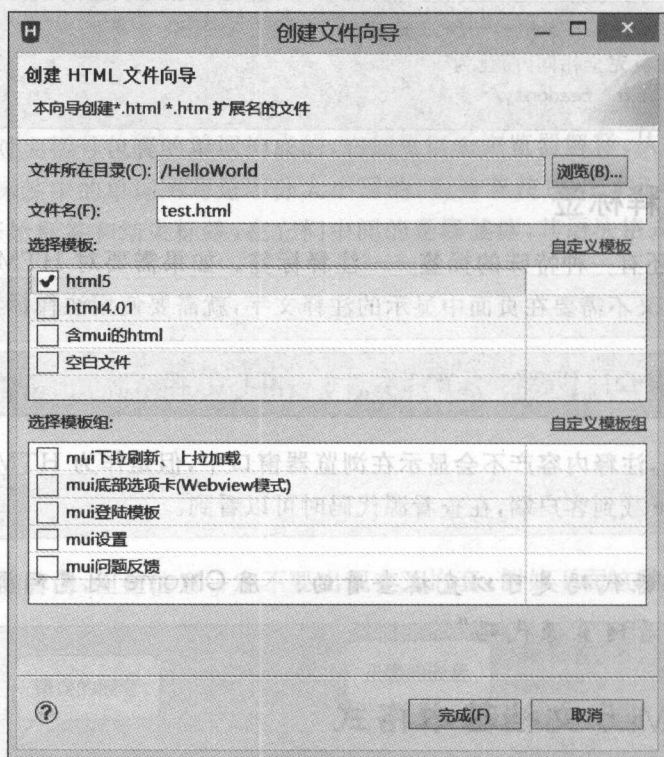


图 2-1 对话框效果

文档类型定义),它必须位于 HTML5 文档中的第一行,也就是位于<html>标签之前。在所有 HTML 文档中规定 DTD 是非常重要的,它告知浏览器当前文档所使用的 HTML 规范,这样浏览器才能了解预期的文档类型。DOCTYPE 不属于标签,它是一条指令。实际移除第一行的 DTD 声明,页面也能浏览,不会报任何错误,但是 HTML 本身就有很多版本,如果希望浏览器能正确无误地支持 HTML5 各种特性,特别是 HTML5 App 开发的页面中,一定要保留这一行。

2. 根标签

<html>标签位于<!DOCTYPE html>声明之后,也作为根标签,用于告知浏览器其自身是一个 HTML 文档。<html>标签标志着 HTML 文档的开始,</html>标签则标志着文档的结束,在它们之间是文档的头部和主体。

3. 头部标签

<head>标签用于定义 HTML 文档的头部信息,它是所有头部元素的容器。<head>中的元素可以引用脚本、指示浏览器在哪里找到样式表、提供元信息等。绝大多数文档头部包含的数据都不会真正作为内容显示出来。

在<head>标签体中,有这样一行 HTML 代码:

```
<meta charset = "UTF - 8">
```

它的作用是告诉浏览器当前页面采用的编码格式是“UTF-8”。UTF-8 是一种字符

编码,除此之外在国内网站常用的还有 GB 2312 和 GBK。GB 2312 和 GBK 主要用于汉字编码,UTF-8 是国际编码,实用性比较强。

<title>标签可以用定义文档的标题。浏览器会以特殊的方式来使用标题,并且通常把它放置在浏览器窗口的标题栏或状态栏上。同样地,当把文档加入用户的链接列表或者收藏夹或书签列表时,标题将成为该文档链接的默认名称。(这个标签对于 HTML5 App 开发的窗口意义不大。)

4. 主体标签

<body>标签用于定义 HTML 文档所要显示的内容,也称为**主体标签**。页面中显示的所有文本、图像、音视频等信息都必须位于<body>标签中,<body>标签中的信息才是最终展示给用户看的。

2.3 布局 div 标签

div 标签作为 HTML 页面中常用的标签,其默认样式是独占一行,它在页面显示时并无任何特殊的显示,它的样式,例如宽度、高度等样式设置、内部字体大小、字体颜色,都需要通过 CSS 来实现。

这个标签的作用就是一个容器,在这个容器中可以放置各种标签内容,主要是通过<div>标签来实现各种各样的布局效果。

<div>标签的宽度在没有使用 CSS 控制时,会自动设置为父容器的宽度,而高度会随着内容进行自适应。这个特性有时在 App 开发中是可以利用的。

【例 2-1】 <div>标签使用示范,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title>div 标签示范</title>
  </head>
  <body>
    <div style = "background - color: greenyellow;">
      div 作为 html 网页中常用的标签,其默认样式是独占一行,其 CSS 样式需要重新赋予. 比如对 div 宽度、高度等样式设置、内部字体大小、字体颜色都需要通过 CSS 来实现.
    </div>
  </body>
</html>
```

为了显示<div>标签的宽度和高度变化,这里特意给它加了灰底,CSS 的使用将在第 3 章进行详细讲解,使用 Chrome 浏览这张页面后,试着改变浏览器的宽度,你会发现该<div>标签的宽度会随着浏览器窗口的宽度进行改变,而高度是随着内容的变化进行自动适应的,如图 2-2 所示。

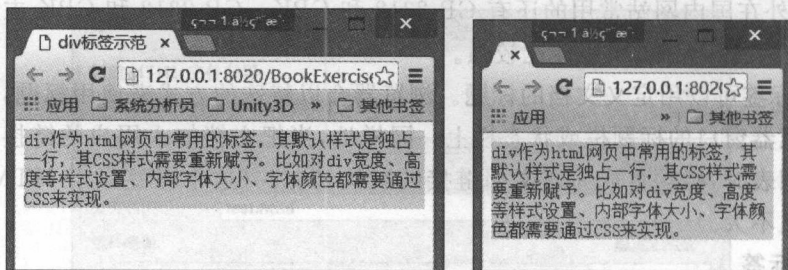


图 2-2 <div>标签的宽度高度自适应性

2.4 文本控制标签

在一个页面中的文字往往占用了较大的篇幅,为了让文字排版整齐、结构清晰,HTML 中提供了一系列的文本控制标签。下面进行详细讲解。

2.4.1 标题 h 标签

为了使页面更有语义化,在页面中会常用到标题标签,HTML 提供了 6 个等级的标题,有<h1>、<h2>、<h3>、<h4>、<h5>、<h6>,其重要性从<h1>~<h6>依次降低,标题中的文字会自动设置为粗体。这个标签的使用见例 2-2,在 Chrome 中运行后的效果如图 2-3 所示。

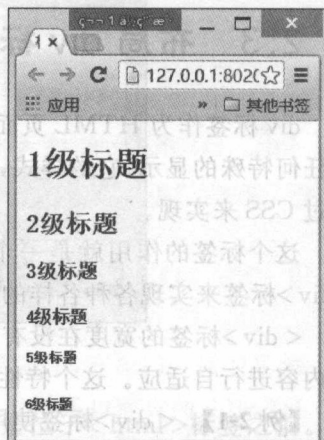


图 2-3 标题标签的使用

【例 2-2】 标题标签使用示范,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title>标题标签示范</title>
  </head>
  <body>
    <h1>1 级标题</h1>
    <h2>2 级标题</h2>
    <h3>3 级标题</h3>
    <h4>4 级标题</h4>
    <h5>5 级标题</h5>
    <h6>6 级标题</h6>
  </body>
</html>
```

2.4.2 段落 p 标签

像写文章一样,在页面中要把文字有条理地显示出来,离不开段落标签,它可以把内容

分为若干段落。段落标签使用<p>,默认情况下,它会根据浏览器窗口的大小自动换行。<p>标签的语法格式为:

```
<p>段落文本</p>
```

和<div>标签一样,<p>标签也会单独占一行,但不同的是,它会自动在段落前后各加一个空行,见例 2-3。

【例 2-3】 段落标签使用示范,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title>段落标签使用示范</title>
  </head>
  <body>
    <p style = "background - color: aquamarine;">HTML5 手机应用的最大优势就是可以在网
    页上直接调试和修改。原先应用的开发人员可能需要花费非常大的力气才能达到 HTML5 的效果,不
    断地重复编码、调试和运行,这是首先得解决的一个问题。因此也有许多手机杂志客户端是基于
    HTML5 标准,开发人员可以轻松调试修改.</p>
    <p style = "background - color: greenyellow;">从性能角度来说,HTML5 首先缩减了 HTML
    文档,使这件事情变得更简单。从用户可读性上说,原先一大堆东西对初学者来说,第一次看到这些
    东西是看不懂的,而 HTML5 的声明方式对用户来说显然更友好一些.</p>
  </body>
</html>
```

为了显示更清晰,在这个例子中使用了 2 个<p>标签,页面启动后的效果如图 2-4 所示。你会发现,段落标签在文字前后都各自添加了一个空行。



和中文的段落不一样,首行不会自动空两格,要实现这个效果,必须借助于 CSS。

2.4.3 水平线 hr 标签

在页面中经常可以使用一些水平线将段落之间隔开,使得文档结构清晰,层次分明。这些水平线可以通过插入图片实现,也可以简单通过水平线标签<hr/>来完成。在页面中输入一个<hr/>,就自动添加了一条默认样式的水平线。例如,在例 2-3 中的源代码的 2 个<p>标签之间加入一个<hr/>标签,会得到如图 2-5 所示的效果。

2.4.4 换行 br 标签

在 HTML 中,一个段落的文字会自动从左到右排列,直到浏览器窗口右端,然后自动换行。如果希望某段文本或标签内容强制换行显示,就需要使用换行标签
,如果只是像在 Word 中编辑时直接按 Enter 键是毫无作用的。

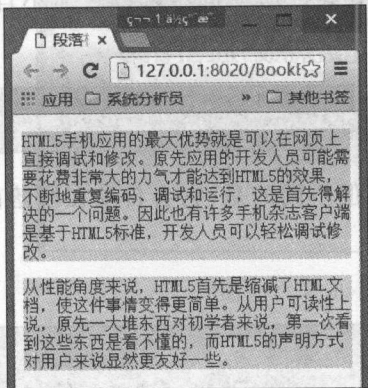


图 2-4 <p>标签的使用



图 2-5 <hr>标签自动加水横线

2.4.5 特殊字符标记

在页面中常常会用到一些包含特殊字符的文本,如数学公式、版权信息等,在 HTML 中为这些字符准备了特殊字符标记,如表 2-1 所示。

表 2-1 常用特殊字符标记

特殊字符	描述	相应的标记
	空格	 或 nbsp;nbsp;
<	小于	<
>	大于	>
&	和	&
©	版权	©
®	注册商标	®
"	双引号	"
²	平方 2	²
°	摄氏度	°

2.4.6 修饰 span 标签

在页面中有时需要为一行文字的某部分使用不同的效果,例如单独设置为红色、粗体、斜体或下画线等,HTML 中有个特殊的< span>标签,可以专门用来定义页面中某些特殊显示的文本,它本身没有任何固定的格式体现,只有应用 CSS 样式时,才会产生视觉上的变化,例如图 2-6 中的效果,这行文字中的“span 标签”是单独的红色,要实现这样的效果,就可以借助于< span>标签。

HTML 中的 span 标签标准用途是什么? - HTML5 - 知乎

图 2-6 文字中的红色文本

用< span>标签加 CSS 就可以轻松实现,文字的 HTML 代码为:

HTML 中的< span style = "color:red"> span 标签标准用途是什么? - HTML5 - 知乎

2.5 图像 img 标签

在 HTML 页面中,任何元素的实现都要依靠 HTML 标签,要想在页面中显示图像,就需要使用图像标签,图像标签的基本语法格式为:

```
<img src = "图像 URL" alt = "文字提示"/>
```

其中,src 属性用来指向图像的 URL 路径,可以使用相对路径,也可以使用网络路径,alt 属性主要用于图像看不到或丢失时显示的文字,另外使用了 alt 属性,也便于 Google 和 Baidu 等搜索引擎的 SEO(Search Engine Optimization,搜索引擎优化),它对于网页有重要作用,便于收录。下面演示了标签的使用。

【例 2-4】 图片标签使用示范,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title>图片标签的使用</title>
  </head>
  <body>
    <!-- 这里图片的 URL 使用的是网络路径 -->
    <img src = "http://www.meishihui68.com.cn/imgs/bd_logo.png"
        alt = "百度的 Logo"/>
    <br />
    <!-- 这里图片的 URL 使用的是相对路径 -->
    <img src = "../img/baidu.png" alt = "百度的 Logo"/>
  </body>
</html>
```

在 Chrome 中浏览后的效果如图 2-7 所示,如果图像路径错误或未设置,则会显示如图 2-8 所示的效果,会显示在 alt 属性中设置的替代文字。



图 2-7 图片标签显示图片



图 2-8 图片不能显示时的效果



在 Chrome 中, 页面上的图片如果是使用标签显示的, 可以把鼠标放在相应图片上, 单击鼠标右键, 可以得到如图 2-9 所示的菜单, 可以通过不同选项对图片作不同的处理。



图 2-9 Chrome 菜单处理图片

2.4.5 特殊字符标记

2.6 超链接 a 标签

在浏览网页时, 经常在单击某标题后, 会打开另一张网页, 这就是使用超链接标签的功劳。超链接标签在网页中占有不可替代的地位, 但是它的使用又非常简单, 只需要使用<a>标签就行了, 基本的语法格式为:

```
<a href = "跳转的 URL 地址" target = "新页面弹出方式">文本或图像</a>
```

其中, href 属性用于指定的链接的目标 URL 地址, 可以使用当前网站内的相对地址, 也可以使用外网地址(但必须加上 http://), target 属性用于指定链接页面的打开方式, 有 _self 和 _blank 两种。默认值为 _self, 表示在原窗口中打开; _blank 表示在新窗口中打开。

除了常规的超链接用法, 在移动设备上使用<a>标签, 还可以用它来实现单击电话号码后, 打开发送短信或拨打电话界面, 也可以用于发邮件。例 2-5 展示了<a>标签的不同用法。

【例 2-5】 超链接标签的不同用法, 代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <title>超链接标签的不同用法</title>
  </head>
  <body>
    <!-- 普通的文字超链接 -->
    <a href = "http://www.163.com" target = "_blank">打开网易</a>
    <a href = "example-2.4.html">打开上个例子</a>
    <br />
    <!-- 图片超链接 -->
    <a href = "http://www.163.com"><img src = "../img/163.png"/></a>
```



```
<br /><br />
<!-- 拨打电话 -->
<a href="tel:10086">拨打电话</a>
<!-- 发送短信 -->
<a href="sms:10086">发送短信</a>
<!-- 发送 Email -->
<a href="mailto:mail@qq.com">发送邮件</a>
</body>
</html>
```

在 Chrome 中浏览这张页面,效果如图 2-10 所示,页面上展示了普通的文字超链接和图片超链接,把这张页面复制到手机上运行,单击对应的超链接也可以打开电话、短信和邮件发送的 App。

2.7 列表标签

为了使页面更易读,经常需要将信息以列表的形式呈现,例如新闻 App 中新闻的呈现的列表,排列有序,内容清晰。为了满足页面排版的需求,这里主要讲解 HTML 语言提供的无序列表和有序列表。

2.7.1 无序列表 ul 标签

无序列表是网页中最常用的列表,之所以称为“无序列表”,是因为其各个列表之间没有顺序级别之分,通常都是并列的,例如淘宝首页中各商品的分类排序不分先后,这就可以看作是一个无序列表。无序列表的基本语法格式为:

```
<ul>
  <li>列表项 1</li>
  <li>列表项 2</li>
  <li>列表项 3</li>
  <li>列表项 4</li>
</ul>
```

在上面的语法中,列表项是使用标签定义的,标签类似于<div>,也是容器,它的标签体也可以包含各种内容,包括文字、图片等。标签只能直接嵌套标签,是不能直接输入文字的。无序列表的显示效果如图 2-11 所示。

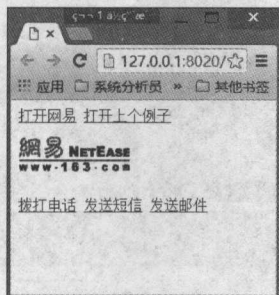


图 2-10 超链接标签的使用

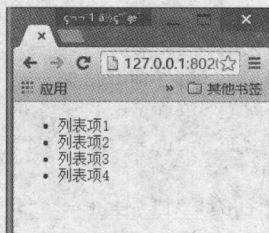


图 2-11 无序列表显示效果

2.7.2 有序列表 ol 标签

有序列表指的是按照字母或数字顺序排列的列表项目。要注意的是,有序列表的结果是带有前后顺序之分的编号,如果插入和删除一个列表项,编号会自动进行调整。无序列表的基本语法格式为:

```
<ol type="项目符号类型值" start="项目符号开始的数值">
  <li>列表项 1</li>
  <li>列表项 2</li>
  <li>列表项 3</li>
  <li>列表项 4</li>
</ol>
```

与无序列表不同的是,列表项的前面可以自动出现字母或数字序号,这是由 type 属性值决定的,type 的取值如表 2-2 所示,start 属性用于指定从第几个字母或数字开始。

表 2-2 无序列表 type 的取值

取值	描 述
1	项目符号用数字表示(1,2,3,...)
A	项目符号用大写字母表示(A,B,C,...)
a	项目符号用小写字母表示(a,b,c,...)
I	项目符号用大写的罗马数字(I,II,III,...)
i	项目符号用大写的罗马数字(i,ii,iii,...)

【例 2-6】 有序列表使用示例的代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>有序列表示例</title>
  </head>
  <body>
    <ol>
      <li>列表项 1</li>
      <li>列表项 2</li>
      <li>列表项 3</li>
    </ol>
    <ol type="a" start="2">
      <li>第 1 项</li>
      <li>第 2 项</li>
      <li>第 3 项</li>
      <li value="20">第 4 项</li>
    </ol>
    <ol type="I" start="2">
      <li>第 1 项</li>
```

```
<li>第 2 项</li>
<li>第 3 项</li>
</ol>
</body>
</html>
```

在 Chrome 中浏览该页面,效果如图 2-12 所示,可以看到有序列表的不同使用方式。

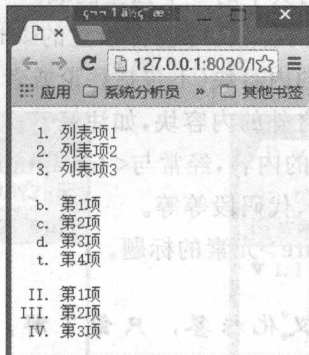


图 2-12 有序列表显示效果

列表项目是可以嵌套的,这意味着在的标签体中,可以再使用或标签。

2.8 语义化标签

在 HTML5 标准出现之前,页面的布局全是使用 div+css 完成的,如图 2-13 所示,这给搜索引擎的 SEO 带来了复杂性,网络蜘蛛(搜索引擎收集网站信息的程序)必须从页面的头读到尾,才能正确收录到需要的信息(例如网站版权信息),而 HTML5 标准制定了许多语义化标签,它可以将每个区域语义化,让页面结构更清晰,更利于 SEO,如图 2-14 所示,这是使用语义化标签后的布局方式。

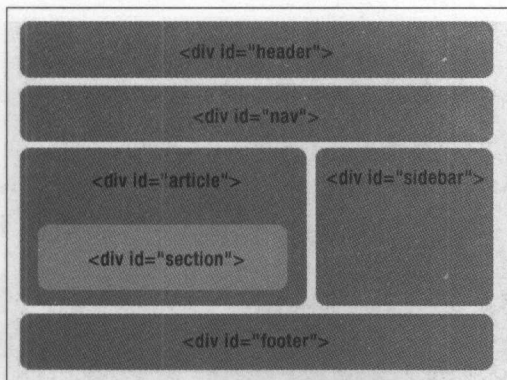


图 2-13 原有的网页布局

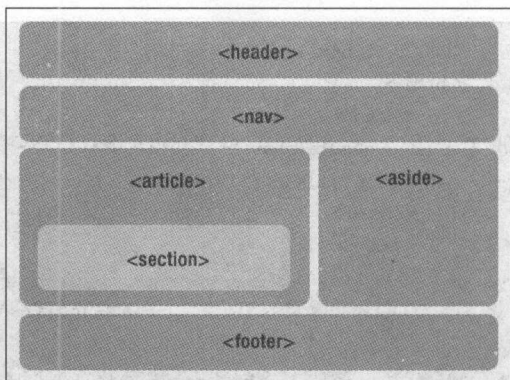


图 2-14 HTML5 页面布局

本书讲解 HTML5 App 开发,重点不是网页设计,所以只简单介绍一些语义化标签,若需要进一步了解,请参考其他书籍。

- `<header>`: 定义网页或文章的头部区域,可包含 logo、导航、搜索条等内容。
- `<main>`: 定义网页中的主体内容。
- `<nav>`: 定义包含多个超链接的区域,标注页面导航链接。
- `<footer>`: 定义网页或文章的尾部区域,可包含版权、备案等内容。
- `<section>`: 通常标注为网页中的一个独立区域。
- `<article>`: 完整、独立的内容块,可包含独立的`<header>`、`<footer>`等结构元素。
如新闻、博客文章等独立的内容块(不包括评论或者作者简介)。
- `<aside>`: 定义除主内容之外的内容块,如注解。
- `<figure>`: 代表一段独立的内容,经常与`<figcaption>`(表示标题)配合使用,可用于文章中的图片、插图、表格、代码段等等。
- `<figcaption>`: 定义`<figure>`元素的标题。



不用太在意这些语义化标签,只需了解,HTML5 App 开发中用得最多的还是`<div>`,也不会涉及 SEO 的问题。

2.9 页面交互性标签

HTML5 不仅增加了许多 Web 页面的特性,为操作新增加了一些对应的交互性标签,让界面体验更好,本节将介绍这些标签。

2.9.1 细节展示 details 和 summary 标签

`details` 标签用于描述文档或文档某个部分的细节。`summary` 标签经常与 `details` 标签配合使用,作为 `details` 标签的第一个子元素,用于为 `details` 定义标题。当单击标题时,可以显示或隐藏 `details` 中的其他内容。例如书籍的目录结构就可以用这两个标签来制作,看下面这个例子。

【例 2-7】 `details` 和 `summary` 标签使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title></title>
  </head>
  <body>
    <details>
      <summary>1.1 HTML5 介绍</summary>
      <ul>
        <li>1.1.1 终将失败的 Flash</li>
        <li>1.1.2 Web 移动应用的未来</li>
```



```
</ul>
</details>
<details>
  <summary>1.2 HTML5 新特性</summary>
</details>
</body>
</html>
```

在 Chrome 中浏览后,效果如图 2-15 所示,完成了一个书籍目录的列表,当单击“1.1 HTML5 介绍”时,会自动展开有关章节,如图 2-16 所示。

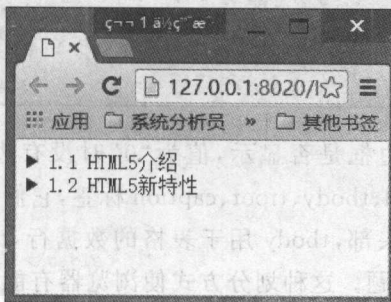


图 2-15 details 和 summary 收缩效果

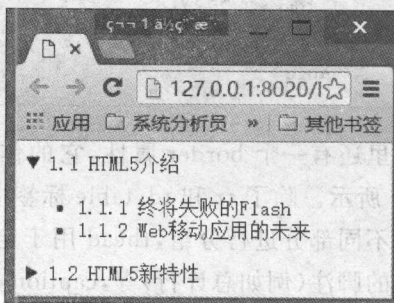


图 2-16 details 和 summary 展开效果

2.9.2 进度条 progress 标签

过去要在页面上完成进度条,基本上是使用 div+css 效果制作的,HTML5 标准中新增了 progress 标签来实现,进度条 progress 标签的语法格式为:

```
<progress value = "当前值" max = "最大值"></progress>
```

需要注意的是: value 和 max 属性的值都必须大于 0,并且 value 的值要小于或等于 max 的值,progress 标签在 Chrome 中的显示效果如图 2-17 所示。

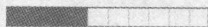


图 2-17 progress 显示效果

2.10 表格标签

表格是页面上经常用来展示数据的一个组件,在 HTML 中提供了 table 标签用于实现表格效果。表格的行使用 tr 标签,列使用 td 标签。如图 2-18 所示,这是一个最简单的存款利率表格,对应的 HTML 代码如下:

```
<table border = "1">
  <tr>
    <td>一年</td>
    <td>1.75</td>
```

```

</tr>
<tr>
  <td>二年</td>
  <td>2.25</td>
</tr>
<tr>
  <td>三年</td>
  <td>2.75</td>
</tr>
<tr>
  <td>五年</td>
  <td>2.75</td>
</tr>
</table>

```

这里还有一个 border 属性,它的作用是设置边框是否显示,值为“0”时没有边框,如图 2-19 所示。除了 tr 和 td,table 标签中还有 thead、tbody、tfoot、caption 标签,它们用于对表格的不同部分进行分组,thead 用于定义表格的头部,tbody 用于表格的数据行,tfoot 用于表格的脚注(例如总计行)等,caption 用于表格标题。这种划分方式使浏览器有能力支持独立于表格标题和页脚的表格正文滚动。当长的表格被打印时,表格的表头和页脚可被打印在包含表格数据的每张页面上。如果要使用 thead、tfoot 以及 tbody 标签,就必须使用全部元素。它们的出现次序是:thead、tfoot、tbody,这样浏览器就可以在收到所有数据前呈现页脚了。

一年	1.75
二年	2.25
三年	2.75
五年	2.75

图 2-18 有边框的表格

一年	1.75
二年	2.25
三年	2.75
五年	2.75

图 2-19 无边框的表格

对于上面的例子,用 thead、tfoot、tbody 进行了一些改造,标题行中的列必须使用 th 标签,代码如下:

```

<table border="1">
  <thead>
    <tr>
      <th>存款时间</th>
      <th>存款利率</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>注:数据来源于中国工商银行</td>
      <td></td>
    </tr>
  </tfoot>

```

```

</tfoot>
<tbody>
  <tr>
    <td>一年</td>
    <td>1.75</td>
  </tr>
  <tr>
    <td>二年</td>
    <td>2.25</td>
  </tr>
  <tr>
    <td>三年</td>
    <td>2.75</td>
  </tr>
  <tr>
    <td>五年</td>
    <td>2.75</td>
  </tr>
</tbody>
</table>

```

在 Chrome 中显示效果如图 2-20 所示,标题行会自动加粗。但这个表格中,三年和五年的存款利率是相同的,让它们的数据合并显示,另外,备注应该跨两格,HTML5 中的表格有两个属性: `colspan` 可以设置横跨列数,而 `rowspan` 可以设置横跨行数。

【例 2-8】 表格 table 的合并示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title>表格以及表格的合并</title>
  </head>
  <body>
    <table border = "1">
      <thead>
        <tr>
          <th>存款时间</th>
          <th>存款利率</th>
        </tr>
      </thead>
      <tfoot>
        <tr>
          <td colspan = "2">注: 数据来源于中国工商银行</td>
        </tr>
      </tfoot>
      <tbody>
        <tr>

```



```
<td>一年</td>
<td>1.75</td>
</tr>
<tr>
<td>二年</td>
<td>2.25</td>
</tr>
<tr>
<td>三年</td>
<td rowspan = "2">2.75</td>
</tr>
<tr>
<td>五年</td>
</tr>
</tbody>
</table>
</body>
</html>
```

在 Chrome 中浏览这个页面后,可以得到如图 2-21 所示的页面效果。

表1.1 存款利率数据表

存款时间	存款利率
一年	1.75
二年	2.25
三年	2.75
五年	2.75
注:数据来源于中国工商银行	

图 2-20 thead 等的使用

表1.1 存款利率数据表

存款时间	存款利率
一年	1.75
二年	2.25
三年	2.75
五年	

注：数据来源于中国工商银行

图 2-21 表格的合并

2.11 表单的应用

表单是页面的重要组成部分,它收集用户的输入信息,并将这些信息发送给服务端程序进行处理。例如注册、登录、查询等功能,如图 2-22 和图 2-23 所示。表单由表单 form 标签和各种输入标签组成,下面分别讲解。

2.11.1 表单 form 标签

在 HTML5 中,<form></form>标签用于定义表单,这个标签在页面上没有显示效果,单独使用也毫无意义。在标签体中的各输入信息将被收集,并向服务器发送。表单的语法格式如下所示:

网易邮箱/常用邮箱

请输入密码

登 录

☐ 十天内免登录 忘记密码?

图 2-22 表单实现的登录

注册字母邮箱 注册手机号码邮箱 注册VIP邮箱

* 手机号码 @163.com
请填写手机号码

* 图片验证码
请填写图片中的字符，不区分大小写 看不清楚? 换张图片
免费获取验证码

* 短信验证码
请查收手机短信，并填写短信中的验证码

* 密码
6~16个字符，区分大小写

* 确认密码
请再次填写密码

☒ 同意“服务条款”和“用户须知”、“隐私权相关政策”

立即注册

图 2-23 表单实现的注册

```
<form method = "提交方式" action = "服务端 url" enctype = "编码方式"></form>
```

对它的属性作相应的说明如下。

- method 属性：用于规定如何发送表单的数据，数据会发送到 action 属性所定义的服务端，它分为 get 和 post 两种方式。

get 方式是默认值，使用这种方法提交的数据将会附加在服务端 url 之后，以?与 url 分开。使用这种方式传输的数据量小，由于受到 url 长度的限制，最多只能传递 1KB。字母数字字符原样发送，空格转换为+，其他符号转换为%XX，其中 XX 为该符号以十六进制表示的 ASCII(或 ISO Latin-1)值。例如 action 服务端的地址是：

```
http://www.example.com/
```

使用 get 方式提交数据 x 和 y 后，浏览器中加载的服务端地址会变成：

```
http://www.example.com/?x=2&y=3
```

post 方式传递的数据量较大，它把数据作为 http 请求的内容，数据不会附加在 url 之后。

- action 属性：指明表单数据要发送到的页面或 API URL，如果这个属性是空的或未写，那么当前的文档 URL 将被使用。
- enctype 属性：可省略，规定在发送到服务器之前应该如何对表单数据进行编码，它的取值见表 2-3。

表 2-3 form 表单的 enctype 属性取值

值	描 述
application/x-www-form-urlencoded	默认值，数据编码为名称/值对
multipart/form-data	将数据内容分段，每段用分隔符隔开，在需要文件上传时，必须设置为该属性
text/plain	空格转换为加号，但不特殊字符编码

2.11.2 各种 input 输入标签

input 标签是表单中最常用的标签，页面中常见的文本框、单选框、复选框等效果都是靠它实现的。input 标签有很多种形式，主要使用 type 属性来设置，下面对它的各种形式进行讲解。

1. 单行文本输入框

单行文本输入框常用于输入各种简短的信息，如账号、用户名、身份证号码等，它的页面显示效果如图 2-24 所示。对应的 HTML 代码为：

```
<input type="text" value="张三丰"/>
```

2. 密码输入框

密码输入框用来输入密码，其内容会以圆点形式显示，对密码自动进行隐藏，它的页面显示效果如图 2-25 所示。对应的 HTML 代码为：

```
<input type="password" value="12345"/>
```

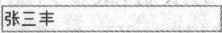


图 2-24 单行文本框

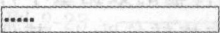


图 2-25 密码输入框

3. 普通按钮

普通按钮在页面显示的效果如图 2-26 所示，这种按钮必须和 JavaScript 代码配合才有作用。对应的 HTML 代码为：

```
<input type="button" value="普通按钮"/>
```

4. 单选框

单选框用于单项选择，如选择性别、是否同意等。要注意的是，在定义单选框时，必须为同

一组的选项指定相同的 name 属性值,这样才能进行单选。另外,可以对它使用 checked 属性,指定选中状态。它在页面中的显示效果如图 2-27 所示。对应的 HTML 代码为:

```
<input type="radio" checked/>男
```

普通按钮

图 2-26 普通按钮

●男

图 2-27 普通按钮

5. 复选框

复选框用于多项选择,如选择兴趣、爱好等。和单选框类似,必须为同一组的选项指定相同的 name 属性值,也可以对它使用 checked 属性,指定选中状态。它在页面中的显示效果如图 2-28 所示。对应的 HTML 代码为:

```
<input type="checkbox" checked/>同意与否
```

6. 提交按钮

当表单中的信息输入完成后,往往需要提交给服务器,在表单中单击提交按钮会自动触发表单的提交动作。可以使用 value 属性修改按钮上的文字,它在页面上显示的效果如图 2-29 所示,从外观上看,与普通按钮没有什么区别,不同的是它会使得表单提交。对应的 HTML 代码为:

```
<input type="submit" value="注册"/>
```

☒ 同意与否

图 2-28 复选框

注册

图 2-29 提交按钮

7. 图片提交按钮

图片提交按钮的功能和提交按钮是一样的,也是单击后触发表单的提交,不同的是它可以使用图片作为按钮。它在页面上显示的效果如图 2-30 所示。对应的 HTML 代码为:

```
<input type="image" src="../../img/micky.jpg"/>
```

8. 重置按钮

如果用户在表单中输入的信息有误,可以使用重置按钮恢复表单的初始状态。它在页面上显示的效果如图 2-31 所示,可以使用 value 属性修改按钮上的文字。对应的 HTML 代码为:

```
<input type="reset" value="取消"/>
```

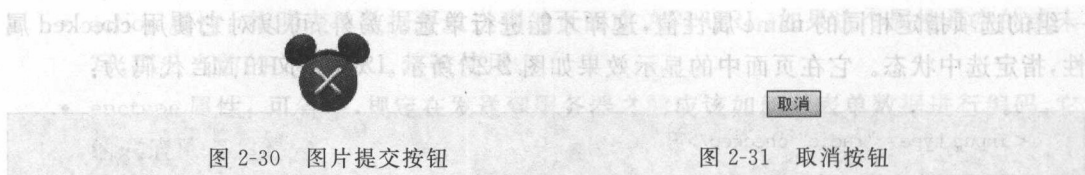



图 2-30 图片提交按钮

图 2-31 取消按钮

9. 隐藏域

正如隐藏域的名称,它在界面上是不显示的,主要用来在表单向服务器提交一些隐藏的数据。对应的 HTML 代码为:

```
<input type = "hidden" value = "这是隐藏的值"/>
```

10. 文件上传域

在页面上经常会需要向服务器提交一些文件,例如照片或文档,这时可以使用文件上传域,它的显示效果如图 2-32 所示,它会出现一个按钮“选择文件”和一个提示“未选择任何文件”,如果单击按钮,就从计算机或移动设备上选取文件后,提示信息会进行相应的改变,在 Chrome 和 Android 中的效果如图 2-33 所示,在 iOS 中的效果如图 2-34 所示。

选择文件 未选择任何文件

图 2-32 文件上传域

选择文件 micky.jpg

图 2-33 在 Chrome 和 Android 中效果

选取文件 1 张照片

图 2-34 在 iOS 中效果

文件上传域还提供了 `multiple` 属性,如果设置了,则用于一次性上传多个文件,它的 HTML 代码格式为:

```
<input type = "file" multiple/>
```

11. Email 输入文本框

显示效果和单行文本输入框完全相同,不同的是,它是专门用于输入 Email 的文本输入框,当表单被提交时,该输入框中的内容会被验证 Email 格式是否正确,如果不正确,会有相应的错误提示信息,对应的 HTML 代码如下:

```
<input type = "email"/>
```

12. URL 输入文本框

显示效果和单行文本输入框完全相同,专门用于 URL 输入的文本框,表单提交时会自动验证 URL 地址格式,如果不正确,会有相应的错误提示信息。对应的 HTML 代码如下:

```
<input type = "url"/>
```

13. 电话输入文本框

显示效果和单行文本输入框完全相同,专门用于输入电话号码,但电话号码格式千差万别,所以它通常会和 `pattern` 属性配合使用(`pattern` 属性在后面讲解),它的 HTML 代码

如下：它在 Windows 中的显示效果如图 2-40 所示，而在 iOS 中的显示效果分别如图 2-40 和图 2-41 所示。

```
<input type="tel"/>
```

14. 关键词搜索文本框

专门用于输入关键词搜索的文本框，在用户输入内容后，其右侧会自动出现一个删除图标，单击这个图标会快速清空输入框。它在页面上的显示效果如图 2-35 所示。它的 HTML 代码如下：

```
<input type="search"/>
```

15. 颜色设置文本框

用于方便用户快速设置颜色的文本框，在过去，要实现颜色选择，必须依靠大量的 JavaScript 代码。它的 HTML 代码如下：

```
<input type="color"/>
```

它在页面上的显示效果如图 2-36 所示，单击后可以进行颜色选择。对于 Chrome，则弹出如图 2-37 所示的颜色选择对话框；对于 Android，则界面效果如图 2-38 所示。很可惜，iOS 上并不支持这个文本框。

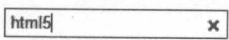


图 2-35 关键词搜索文本框效果



图 2-36 颜色设置文本框

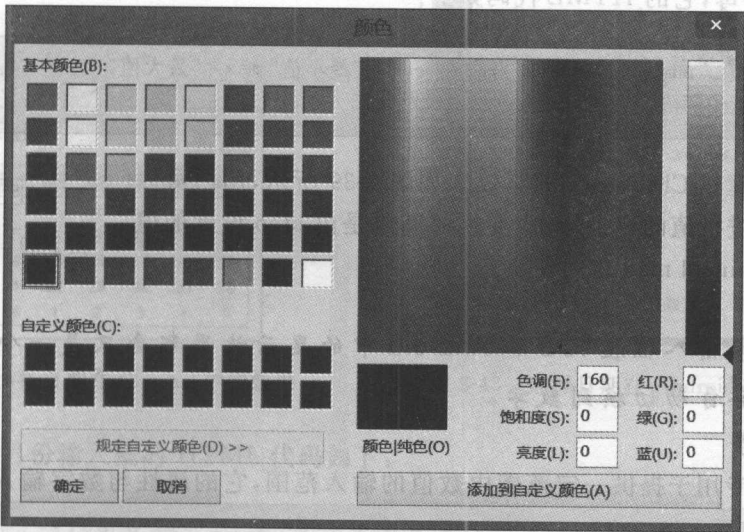


图 2-37 颜色选择对话框

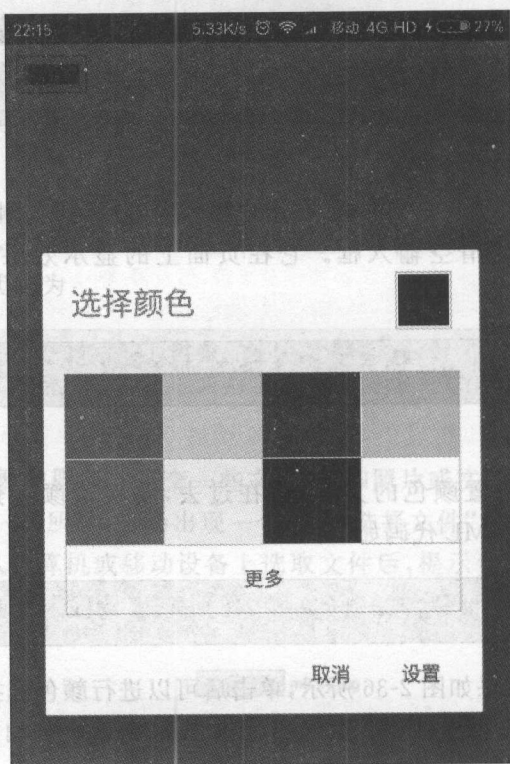


图 2-38 Android 上的颜色设置框

16. 数字输入框

数字输入框是专用于输入数字的,并且还可以对输入的数字进行限制,它只能输入数字,不能输入字母,它的 HTML 代码如下:

```
<input type="number" value="当前值" min="最小值" max="最大值"
step="值的间隔"/>
```

这个输入框在 Chrome 中浏览效果如图 2-39 所示,用户可以单击上下按钮进行值的变换,每次变换的间隔是由 step 设置的值决定的,值在 min 和 max 之间变换。



图 2-39 数字输入框



数字输入框在 iOS 和 Android 中的显示效果都会只是一个文本框,只不过输入法会自动切换到数字。

17. 滑动条

滑动条是专用于提供一定范围内数值的输入范围,它的属性与数字输入框是类似的,HTML 代码如下:

```
<input type="range" value="当前值" min="最小值" max="最大值"
step="值的间隔"/>
```


它在 Windows 中的 Chrome、Android 和 iOS 中的显示效果分别如图 2-40 和图 2-41 所示。

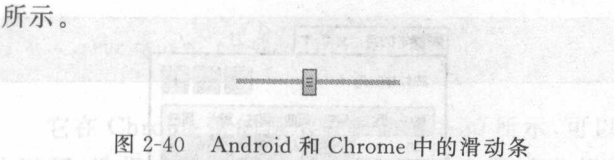


图 2-40 Android 和 Chrome 中的滑动条

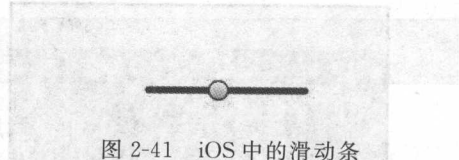


图 2-41 iOS 中的滑动条

18. 日期和时间输入框

HTML5 中提供了多个可供日期和时间选择的输入框,这在以前,必须依靠大量的 JavaScript 代码或 jQuery 插件才能实现。在不同的操作系统下,这些输入框还能显示出不同的外观形式。

(1) 选择日期输入框的 HTML 代码如下:

```
<input type="date"/>
```

在 Chrome、Android、iOS 中的效果分别如图 2-42、图 2-43 和图 2-44 所示。

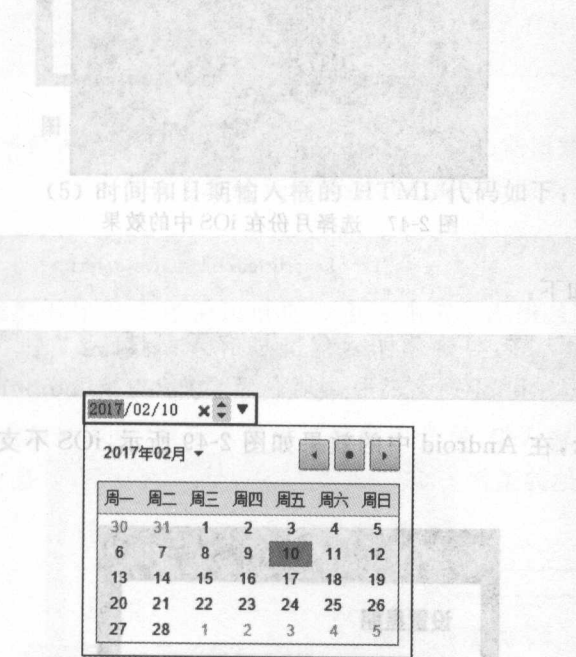


图2-42 选择日期在 Chrome 中的效果



图 2-43 选择日期在 Android 中的效果

(2) 选择月份输入框的 HTML 代码如下:

```
<input type="month"/>
```

它在 Windows 的 Chrome 中效果和选择日期输入框的效果类似,只是选择后只会填入相应的月份,如图 2-45 所示,在 Android、iOS 中的效果分别如图 2-46 和图 2-47 所示。

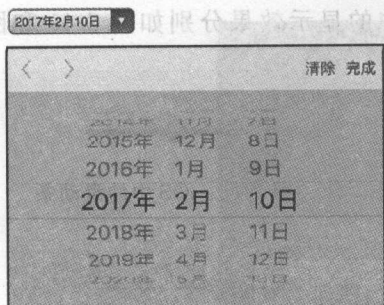


图 2-44 选择日期在 iOS 中的效果



图 2-45 选择月份在 Chrome 中的效果

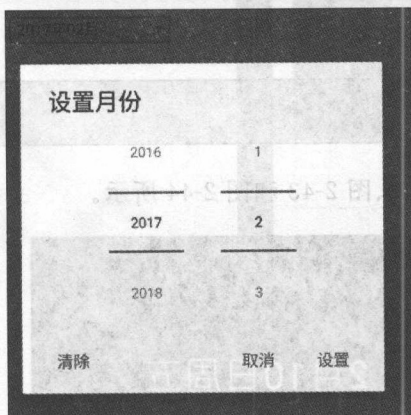


图 2-46 选择月份在 Android 中的效果

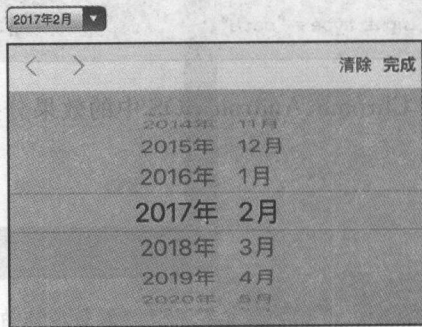


图 2-47 选择月份在 iOS 中的效果

(3) 选择星期输入框的 HTML 代码如下：

```
<input type="week"/>
```

它在 Chrome 中的效果如图 2-48 所示，在 Android 中的效果如图 2-49 所示，iOS 不支持选择星期。



图 2-48 选择星期在 Chrome 中的效果

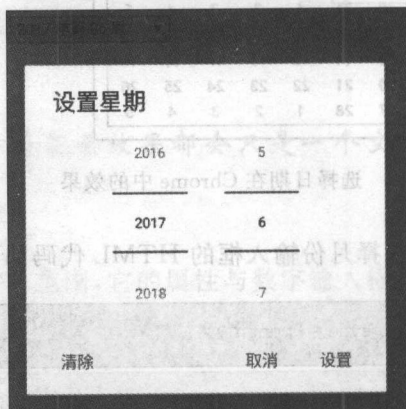


图 2-49 选择星期在 Android 中的效果

图 2-50 (4) 时间输入框的 HTML 代码如下:

```
<input type="time"/>
```

它在 Chrome 中的显示效果如图 2-50 所示,可以手工输入时间,选择小时或分钟后,还可以用上下键进行调整,在 Android 和 iOS 中的效果分别如图 2-51 和图 2-52 所示。

11:00 x

图 2-50 时间输入在 Chrome 中的显示效果

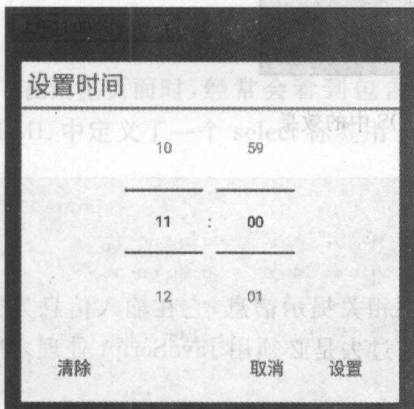


图 2-51 时间输入在 Android 中的效果

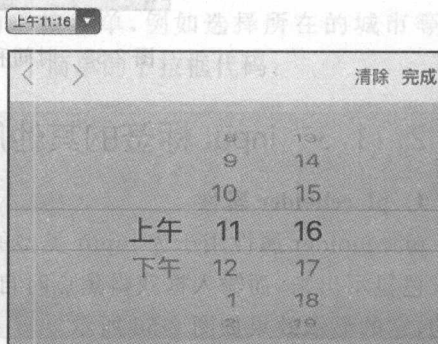


图 2-52 时间输入在 iOS 中的效果

(5) 时间和日期输入框的 HTML 代码如下:

```
<input type="datetime-local"/>
```

它是日期输入和时间输入的综合体,在 Chrome 中的显示效果如图 2-53 所示,在 Android 和 iOS 的效果分别如图 2-54 所示和图 2-55 所示。



图 2-53 时间和日期输入在 Chrome 中的效果

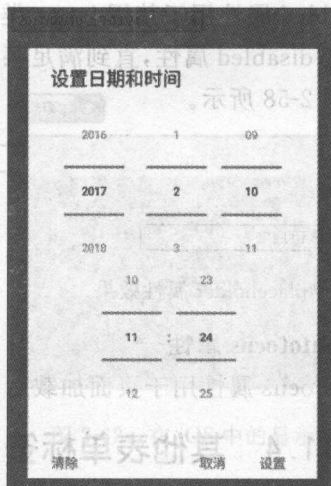


图 2-54 时间和日期输入在 Android 中的效果

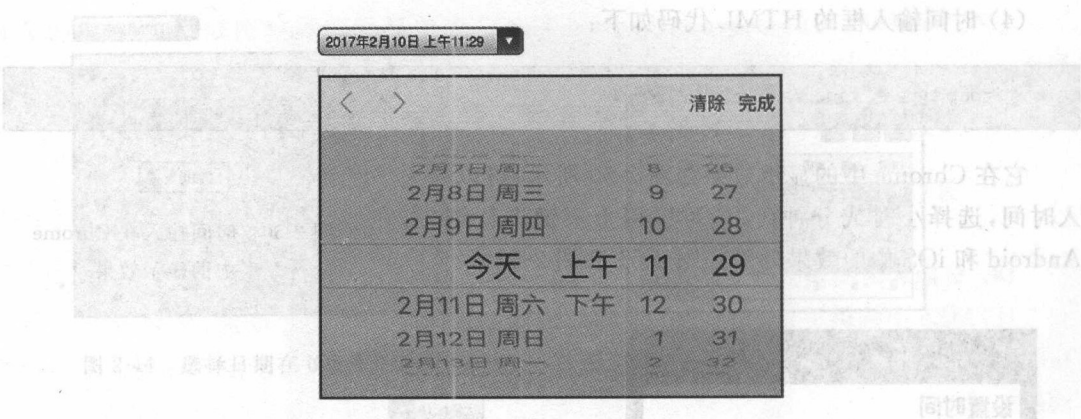


图 2-55 时间和日期输入在 iOS 中的效果

2.11.3 input 标签的其他属性

1. placeholder 属性

placeholder 属性用于为 input 类型的输入框提供相关提示信息，它在输入信息为空时以灰色显示出来，而输入框获得焦点时自动消失，这在过去是必须用 JavaScript 处理才能实现的，它的显示效果如图 2-56 所示。

2. required 属性

required 属性用于在 form 表单提交数据前，为 input 类型的输入框规定必须填入数据，如果未输入，则有错误信息提示（iOS 并不支持这个属性），如图 2-57 所示。

3. pattern 属性

pattern 属性用于在 form 表单提交数据前，为 input 类型的输入框规定数据必须符合正则表达式，如果格式不正确，则有错误信息提示。正则表达式的知识在第 4 章中会详细介绍。

4. disabled 属性

disabled 属性用于禁用 input 类型的输入框，被禁用的输入框既不可用，也不能单击。可以设置 disabled 属性，直到满足某些其他的条件为止（例如选择了一个复选框等）。显示效果如图 2-58 所示。

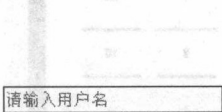


图 2-56 placeholder 属性效果

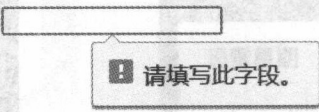


图 2-57 required 属性效果

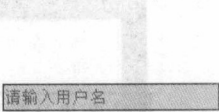


图 2-58 disabled 属性效果

5. autofocus 属性

autofocus 属性用于页面加载后，为 input 类型的输入框自动获取光标焦点。

2.11.4 其他表单标签

1. textarea 标签

当需要输入大量文本信息时，单行文本输入框不太适用，在 HTML 中可以通过

textarea 标签轻松地创建多行文本输入框。

它的大小最好使用 CSS 来定义,另外,在 Chrome 下浏览,文本框的大小是可以使用鼠标拖拉控制的,效果如图 2-59 所示。

它的 HTML 代码如下:

```
<textarea>
  文字内容在这里!
</textarea>
```

2. select 标签

在浏览页面时,经常会看到包含多个选项的下拉菜单,例如选择所在的城市等。在 HTML 中定义了一个 select 标签用于实现,这是一个简单的下拉框代码:

```
<select>
  <option value="item1">选项 1</option>
  <option value="item2" selected>选项 2</option>
  <option value="item3">选项 3</option>
  <option value="item4">选项 4</option>
</select>
```

它在 Chrome 中的浏览效果如图 2-60 所示,在 Android 和 iOS 中的显示效果分别如图 2-61 和图 2-62 所示。

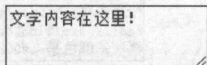


图 2-59 textarea 标签效果

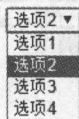


图 2-60 在 Chrome 中的浏览效果



图 2-61 在 Android 中的显示效果

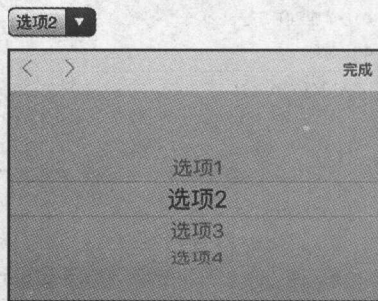


图 2-62 在 iOS 中的显示效果

select 标签中的每个选项是使用 option 标签定义的,< option >和</option >之间定义显示的选项文字,value 属性表示选项的值(form 表单提交数据时,提交的是 value 属性值)。

如果 value 属性没有定义,则该选项的值默认为当前文字。selected 属性可以设置用于被选中的选项,未设置则默认第一个选项被选中。



在表单中要提交数据的输入标签,必须为其设置 name 属性值,这是最容易忽略的地方。如果没有设置,则该输入标签的数据无法传递。

2.11.5 实例：注册表单

下面综合利用所学的表单和各种 input 输入标签来创建一个常见的注册表单,并将数据提交到服务端: <http://www.meishihui68.com.cn/FormTest.ashx>,由于服务端程序已经固定了,所以各输入标签的 name 属性请按示例固定书写。界面没有美化,将在下一章使用 CSS 技术为其定制外观。

【例 2-9】 注册表单,综合应用 form 和各输入标签,界面效果如图 2-63 所示,数据正确提交给服务端后,页面显示如图 2-64 所示。

图 2-63 注册界面效果

用户名是: huangbo
密码是: 1234
省份是: sichuan
性别是: boy
年龄是: 20
生日是: 2017-02-09
电话是: 13888888888
头像文件名是: micky.jpg
个人主页: http://www.baidu.com
Email: asdf@163.com
喜欢的颜色是: #008040

图 2-64 数据提交后页面显示

界面的 HTML 源代码示范如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title>注册界面: 表单和各输入框综合应用</title>
  </head>
  <body>
    <form action = "http://www.meishihui68.com.cn/FormTest.ashx"
    method = "post" enctype = "multipart/form - data">
      <div>
        <span>用户名: </span>
        <input type = "text" placeholder = "请输入用户名"
          required name = "uname" />
        <input type = "button" value = "检查用户名是否存在" />
```



```

</div>
<div>
    <span>密码: </span>
    <input type="password" placeholder="请输入密码"
        required name="upass" />
</div>
<div>
    <span>确认密码: </span>
    <input type="password" placeholder="请确认密码" required/>
</div>
<div>
    <span>区域: </span>
    <select name="uprov">
        <option value="sichuan">四川省</option>
        <option value="hunan">湖南省</option>
        <option value="guangdong">广东省</option>
    </select>
</div>
<div>
    <span>性别: </span>
    <input type="radio" name="ugender" checked value="boy"/>男
    <input type="radio" name="ugender" value="girl"/>女
</div>
<div>
    <span>年龄: </span>
    <input type="number" name="uage" value="20"
        min="20" max="30" step="1" />
</div>
<div>
    <span>生日: </span>
    <input type="date" name="udate" />
</div>
<div>
    <span>手机号: </span>
    <input type="tel" name="utel" required/>
</div>
<div>
    <span>头像: </span>
    <input type="file" name="uphoto" />
</div>
<div>
    <span>主页: </span>
    <input type="url" name="uurl" required/>
</div>
<div>
    <span>Email: </span>
    <input type="email" name="uemail" required/>
</div>
<div>

```

```

        <span>喜欢的颜色: </span>
        <input type = "color" name = "ubackcolor" />
    </div>
    <div>
        <input type = "checkbox" checked/>同意服务条款
    </div>
    <div>
        <input type = "submit" value = "注册"/>
        <input type = "reset" value = "取消"/>
    </div>
</form>
</body>
</html>

```

2.12 移动开发中 meta 标签的应用

HTML5 移动开发中的一些 meta 专属头部标签,能够帮助浏览器更好地解析 HTML 代码,从而为 HTML5 移动开发提供更好的前端表现与体验。

1. 控制页面的缩放

在 HTML5 App 中的头部标签的标签体中,通常需要加入下面一行代码以控制页面的缩放,让页面自动适应设备的宽度。

```

<meta name = "viewport" content = "width = device - width, initial - scale = 1.0, maximum -
scale = 1.0, minimum - scale = 1.0, user - scalable = no" />

```

meta 标签有很多用法,包括针对搜索引擎和更新频度的描述和关键词等,在用于移动页面缩放控制时,必须使用属性 name="viewport"(viewport 是用来显示页面的那一块区域),另外,content 中的参数含义如下:

- width: 控制 viewport 的大小,device-width 为设备的宽度。
- initial-scale: 页面初始缩放程度,一个浮点值,是页面大小的一个乘数。
- maximum-scale: 页面最大缩放程度,一个浮点值,用以指出页面大小与屏幕大小相比的最大乘数。
- minimum-scale: 页面最小缩放程度,一个浮点值,用以指出页面大小与屏幕大小相比的最大乘数。
- user-scalable: 用户是否能改变页面缩放程度,如果设置为 yes,则允许用户对其进行改变,反之为 no。默认值是 yes。

对于 HTML5 App 中的页面,如果没有这个 meta 标签控制缩放,移动设备会自动重置文字大小,页面可能会模糊不清,而使用 meta 标签控制好缩放,字体大小不会被重置,如图 2-65 和图 2-66 所示。

2. iOS 系统的一些控制

针对 iOS 系统,meta 标签可以控制全屏、状态栏颜色、主屏标题等内容,设置如下:

```
<!-- 强制全屏 -->
<meta name="Apple-mobile-web-App-capable" content="yes" />
<!-- 设置状态栏颜色 -->
<meta name="Apple-mobile-web-App-status-bar-style" content="black" />
<!-- 设置添加至主屏标题 -->
<meta name="Apple-mobile-web-App-title" content="标题" />
```

苍峦滴翠，云烟茫茫；红日溢血，奇石兀立；钟馨琴音，空谷传响——那是昆仑三胜何足道正高山抚琴。这位何足道之风趣自谦在此绰号中展露无疑，可能还有几分兴味阑珊的自嘲之意。他身着白衣，样貌清雅，满腹才华却曲高和寡，以至于划地为局，独自对弈。郭襄生于战火纷飞之中，平生际遇堪称一绝，奔忙俗务的时候多，于诗词琴曲恐怕见解不深。而这位颇为清高琴中高手何足道居然乐意与郭襄切磋琴艺，畅谈曲词，这一点颇值得玩味。郭襄的豪爽大气，为人和善恐怕才是吸引这位异人的原因，从此如郭襄之遇杨过一般眷恋难舍，为这个少女谱了一首新曲以表心意。可惜郭襄痴恋杨过志不在此，何足道也只好带着朦胧的情愫回到昆仑，幸而赠曲之事已成，否则更要引以为恨了吧。张君宝原是觉远身边的小和尚，无意间练成功夫却又被诬陷偷学而被逐出师门。郭襄是伤心人无疑，但见到张君宝罹此大难，暂时搁置下了天涯思君的追寻，倾全力帮扶他。她把金丝镯送给张君宝要他去襄阳找靖蓉夫妇，教导他要有独立的人格，尚不忘提醒他自己有个蛮横难相处的姐姐，郭襄的热忱和善心真正秉承了郭家一脉相承的侠义情怀。这个大他几岁的姐姐在小和尚的一生中起到了不可或缺的作用，鞭策着他开创与兴盛了武当派，从此成为一代宗师。

图 2-65 使用 meta 的显示效果

苍峦滴翠，云烟茫茫；红日溢血，奇石兀立；钟馨琴音，空谷传响——那是昆仑三胜何足道正高山抚琴。这位何足道之风趣自谦在此绰号中展露无疑，可能还有几分兴味阑珊的自嘲之意。他身着白衣，样貌清雅，满腹才华却曲高和寡，以至于划地为局，独自对弈。郭襄生于战火纷飞之中，平生际遇堪称一绝，奔忙俗务的时候多，于诗词琴曲恐怕见解不深。而这位颇为清高琴中高手何足道居然乐意与郭襄切磋琴艺，畅谈曲词，这一点颇值得玩味。郭襄的豪爽大气，为人和善恐怕才是吸引这位异人的原因，从此如郭襄之遇杨过一般眷恋难舍，为这个少女谱了一首新曲以表心意。可惜郭襄痴恋杨过志不在此，何足道也只好带着朦胧的情愫回到昆仑，幸而赠曲之事已成，否则更要引以为恨了吧。张君宝原是觉远身边的小和尚，无意间练成功夫却又被诬陷偷学而被逐出师门。郭襄是伤心人无疑，但见到张君宝罹此大难，暂时搁置下了天涯思君的追寻，倾全力帮扶他。她把金丝镯送给张君宝要他去襄阳找靖蓉夫妇，教导他要有独立的人格，尚不忘提醒他自己有个蛮横难相处的姐姐，郭襄的热忱和善心真正秉承了郭家一脉相承的侠义情怀。这个大他几岁的姐姐在小和尚的一生中起到了不可或缺的作用，鞭策着他开创与兴盛了武当派，从此成为一代宗师。

图 2-66 不使用 meta 的显示效果

3. 控制是否自动识别电话号码和 Email

对于 HTML5 App 中的页面，如果不想让系统自动识别页面中出现的电话号码或 Email 格式数据，可以使用 meta 标签作进行设置：

```
<meta name="format-detection" content="telephone=no, email=no"/>
```

小结

本章主要介绍 HTML5 App 开发中页面内容的定制方式——HTML 语言的一些特点以及 HTML5 页面的文档结构；学习了 HTML 语言中的布局 div 标签、各种文本控制标签、图像标签、超链接标签、列表标签、语义化标签、页面交互性标签、表格标签、表单标签、各种 input 输入标签等；针对 App 开发，还讲解了 meta 标签在移动应用开发中的应用。标签学习难度都不大，重在熟练，需要读者重视并多练习。

习题

一、选择题

1. 下列()项是换行符标签。

- A. hr B. br C. p D. span
2. 以下标签中用于设置页面标题的是()。
- A. html B. body C. head D. title
3. 以下有关列表的说法中,错误的是()。
- A. 有序列表和无序列表可以互相嵌套。
- B. 指定嵌套列表时,也可以具体指定项目符号或编号样式。
- C. 无序列表应使用 ul 和 li 标签进行创建。
- D. li 标签体中不能再出现 ul 标签。
4. 如果要在表单里创建一个单行文本输入框,以下写法中正确的是()。
- A. <input/> B. <input type="text"/>
- C. <input type="password"/> D. <input type="tel"/>
5. 在指定单选框时,只有将以下()属性的值指定为相同,才能使选项成为一组。
- A. type B. name C. value D. checked
6. 以下有关表单中按钮的说法中,错误的是()。
- A. 可以用图像作为提交按钮
- B. 可以用图像作为重置按钮
- C. 可以控制提交按钮上的显示文字
- D. 可以控制重置按钮上的显示文字
7. 在 HTML5 页面中插入半角的大于号">",使用的标记符应该是()。
- A. < B. > C. & D. °
8. 在 form 表单中,看不到的 input 标签是()。
- A. <input type="file"/> B. <input type="hidden"/>
- C. <input type="password"/> D. <input type="reset"/>
9. 表格的()属性可以使相邻行进行合并。
- A. cellpadding B. cellspacing C. colspan D. rowspan
10. 在下列的 HTML 代码中,()项可以产生超链接。
- A. meishihui68
- B. meishihui68
- C. meishihui68
- D. meishihui68

二、判断题

1. div 默认的宽度是由其里面的内容决定的。()
2. 以下使用 a 标签链接到 Baidu 网站的代码是正确的。()

```
<a href="www.baidu.com">Baidu</a>
```

3. 所有的 HTML 标签都有开始标签和结束标签。()
4. HTML 表格在默认情况下有边框。()
5. input 标签中的 required 属性可以用于在表单提交数据时不能为空。()

6. 上传文件时,表单 form 的 enctype 属性设置为 multipart/form-data。()

7. HTML 书写错误时,浏览器会弹出报错信息。()

三、填空题

1. HTML 中设置文档的正文部分的开始标签是_____ ; 结束标签是_____。

2. 在页面中实现下拉框,可以使用_____ 标签和_____ 标签实现。

3. 在页面为 input 输入标签设置提示信息应使用_____ 属性,当它获取焦点时,提示信息自动消失。

4. 可以使用_____ 标签定义表格。

5. HTML 标签的注释可以使用_____。

四、简答题

1. 解释 App 开发中,head 标签中下面这行代码中参数的各含义:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0, user-scalable=no" />
```

2. HTML5 中的语义化标签作用是什么?

3.2 CSS 核心基础

CSS 即层叠样式表(Cascading StyleSheet)。在页面制作时采用 CSS 技术,可以有效地提高网页的美观性,使网页更具吸引力。CSS 技术是网页设计的重要组成部分,它与 HTML 技术共同构成了网页设计的基础。CSS 技术可以实现对网页元素的精确控制,包括颜色、字体、大小、位置等。通过 CSS,可以实现网页的布局、美化,使网页更具吸引力。CSS 技术还可以实现网页的响应式设计,使网页在不同设备上都能正常显示。CSS 技术是网页设计的重要工具,也是网页设计人员必须掌握的技能。

在 CSS 中,selector 代表 CSS 选择器,property 代表 CSS 属性,value 代表的是 CSS 属性对应的值。图 3-2 是一个典型的 CSS 定义。在 CSS 中,selector 的作用是选择 HTML 元素,property 的作用是设置元素的属性,而 value 则是属性的值。CSS 的定义通常由 selector、property 和 value 三部分组成。例如,selector 可以是元素名、类名、ID 名等,property 可以是 color、font-size、text-align 等,而 value 则是具体的属性值,如 red、14px、center 等。CSS 的定义可以应用于单个元素,也可以应用于多个元素。通过 CSS,可以实现对网页元素的精确控制,使网页更具吸引力。

图 3-2 一个典型的 CSS 定义

第3章

CHAPTER 3

CSS 样式设计

学习目标

- 熟练掌握 CSS 的样式规则, CSS 在 HTML5 页面中的应用和各种 CSS 选择器的使用, CSS 的层叠性和优先级。
- 熟练掌握各种 HTML5 App 开发常用的 CSS 属性和页面的适配。
- 掌握使用 Chrome 的“开发者工具”对 CSS 样式进行调试。

CSS 样式设计是 HTML5 App 开发中最重要的技术之一,有了它才真正实现了内容与外观的分离,通过它可以控制页面的布局、样式、动画,移动设备的适配。目前 CSS 也是各公司 HTML5 工程师必备的技能之一。本章针对 CSS 的语法规则、各种在 App 开发中常用的 CSS 属性、CSS 在 Chrome 中的调试等重要内容作详细的讲解。

3.1 CSS 简介

CSS 即层叠样式表(Cascading StyleSheet)。在页面制作时采用 CSS 技术,可以有效地对页面的布局、字体、颜色、背景,甚至动画效果实现精确的控制。只要对相应的代码做一些简单的修改,就可以改变同一页面的外观。CSS 禅意花园(<http://www.csszengarden.com/>)是网站设计领域最著名的网站之一,网站提供了一张 HTML 页面,设计师们为它设计出成百上千个 CSS 样式文件,这张页面通过更换样式表呈现出各式各样、另人惊叹的效果,如图 3-1 所示,这两张页面的源码是一样的,只是样式表文件不同,这让人不禁感叹 CSS 的强大。

在页面中使用 CSS 技术,可以设计出更加整洁、漂亮的页面,它解决了内容与外观分离的问题。科学地编写 CSS,还可以大大提高页面样式的复用性。

CSS 目前的最新版本是 CSS3,由于各浏览器厂商对 CSS3 的各属性支持程度不一样,因此,有少数 CSS3 属性需要用厂商的前缀加以区分,通常把这些加上私有前缀的属性叫“私有属性”,以便于在不同的浏览器下更好地体验 CSS3 特性。表 3-1 列举了各主流浏览器的私有属性。



图 3-1 禅意花园的不同 CSS 设计

表 3-1 各主流浏览器私有属性

内核	浏览器	私有前缀
Trident	IE8/IE9/IE10/IE11	-ms-
Webkit	Chrome/Safari	-webkit-
Gecko	FireFox	-moz-
Presto	Opera	-o-

当一个 CSS3 属性成为标准属性,并且被主流浏览器普遍兼容的时候,就可以省略私有前缀了。

3.2 CSS 核心基础

3.2.1 CSS 样式规则

使用 HTML 时,需要遵守一定的规范,CSS 也是如此。要想熟练地应用 CSS 进行页面样式设计,首先需要了解 CSS 的样式规则。CSS 定义的语法格式如下:

```
selector {property1: value1;property2: value2;...}
```

其中,selector 代表 CSS 选择器,property 代表 CSS 属性,value 代表的是 CSS 属性对应的值,图 3-2 是一个典型的 CSS 定义,它的作用是将 h1 标签内的文字颜色设置为红色,字体大小设为 14 像素。



图 3-2 一个典型的 CSS 定义

3.2.2 CSS 中的单位和颜色

1. 单位

在 HTML5 App 开发中常用的单位及说明如表 3-2 所示。

表 3-2 CSS 单位及说明

单位	描 述
%	百分比,以父元素的大小计算
em	通常 1em=16px,2em=32px,当用于指定字体大小时,em 单位是指父元素的字体大小
ex	相对于字符 x 的高度。此高度通常为字体尺寸的一半
px	像素,是屏幕上显示数据的最基本的点
rem	相对单位,相对 html 标签,常用于 HTML5 页面自适应

2. 颜色

在 HTML5 页面开发过程中经常涉及颜色设置,例如字体颜色、背景色等,颜色的设置可以使用表 3-3 中的方式。

表 3-3 CSS 颜色设置方式及说明

方 式	描 述
预定义颜色名	例如 red、black、blue 等
rgb(x,x,x)	红绿蓝值,例如 rgb(255,234,244)
rgba(x,x,x,a)	红绿蓝透明度值,例如 rgba(255,234,244,0.5)
#rrggbb	十六进制数,例如 #ff0000
HSL	色调(Hue)、饱和度(Saturation)、亮度(Lightness)三个颜色通道的改变以及它们相互之间的叠加来获得各种颜色,Hue 取值范围为 0~360,0(或 360)表示红色,120 表示绿色,240 表示蓝色,也可取其他数值来指定颜色。Saturation(饱和度)取值为:0~100.0%。Lightness(亮度)取值为:0~100.0%,例如 hsl(120,65%,75%)
HSLA	HSL 颜色值的扩展,带有一个 Alph 通道——它规定了对象的不透明度。例如 hsla(120,65%,75%,0.3)

3.2.3 在 HTML 文档中应用 CSS

要想使用 CSS 修饰页面,就需要在 HTML 页面中引入 CSS 样式表。引入 CSS 样式的常用方式有 3 种,具体如下。

1. 内联样式

内联样式是指通过 HTML 标签的 style 属性来设置标签的样式,示例如下:

```
<div style="color:red;font-size:14px;">HTML5 App 开发</div>
```

2. 内嵌样式

内嵌样式是指将在 HTML5 文档的 head 标签体中增加一个< style></style>标签,将 CSS 设置集中在 style 的标签体中定义,基本的语法格式如下:

```
<head>
  <style>
    selector {property1: value1;property2: value2;...}
  </style>
</head>
```

3. 链接样式

链接样式是指将 CSS 样式定义在一个或多个以 CSS 为扩展名的外部样式文件中,通过 head 标签体中使用 link 标签将外部样式表文件链接到 HTML5 页面中,这也是页面样式复用经常会用到的方式,基本的语法格式如下:

```
<head>
  <link rel="stylesheet" href="CSS 文件路径" />
</head>
```

3.3 CSS 选择器

要想将 CSS 样式应用于特定的 HTML 标签,首先需要找到该目标标签。在 CSS 中,执行这一任务的样式规则称为选择器。除了内联样式,内嵌样式和链接样式都需要设计选择器。下面具体介绍这些选择器。

3.3.1 基础选择器

1. 标签选择器

标签选择器指的是用 HTML 的标签名作为选择器,所有标签名都可以作为标签选择器使用,它用于为页面中某一种标签指定统一的 CSS 样式,但这也是缺点,不能实现同一种标签设计的差异化。它的语法示例为:

```
p{font-size:12px;color:red}
```

这就为页面中的所有段落的文字设计了样式:字体大小为 12 像素,颜色为红色。

2. id 选择器

id 选择器使用“#”进行标识,后面紧跟 HTML 标签的 id 属性值,一张页面中的 HTML 标签的 id 属性值是唯一的,所以这种选择器设计的样式只能针对 HTML 页面中某一个具体的标签。例如下面的样式:

```
#mydiv{font-size:12px;color:red}
```

页面启动后,该样式会自动应用到下面的标签元素上:

```
<div id="mydiv">
```



```
HTML5 App 开发
</div>
```

3. 类选择器

类选择器使用“.”(英文点)进行标识,后面紧跟 HTML 标签的 class 属性值。它最大的优势是可以为具有相同 class 属性的 HTML 标签设置相同的样式。例如下面的样式:

```
.myclass{font-size:12px;color:red}
```

页面启动后,该样式会自动应用到 HTML 页面中 class 属性为 myclass 的所有 HTML 标签上,例如下面的 HTML 代码中,div 和 p 标签的 class 属性都是 myclass,它们内部的文字大小都是 12 像素,颜色为红色。

```
<div class="myclass">HTML5 App 开发</div>
<p class="myclass">HTML5 已经于 2014 年 10 月正式定稿</p>
```

4. 限定式选择器

限定式选择器由两个选择器构成,其中第一个为标签选择器,第二个为类选择器或 id 选择器,中间是没有空格的,例如下面的选择器:

```
div #mydiv{...} //为 id 属性为"mydiv"的 div 标签设计样式
p.myclass{...} //为 class 属性为"myclass"的 p 标签设计样式
```

5. 后代选择器

后代选择器是用来选择 HTML 标签元素的后代的,其写法是把父标签的选择器写在前面,后代标签的选择器写在后面,两者之间有一个空格。例如下面的选择器:

```
div p{...} //为 div 标签中的 p 标签设计样式
div #mydiv{...} //为 div 标签中的 id 属性为 mydiv 的子标签设计样式
p.myclass{...} //为 p 标签中 class 属性为 myclass 的子标签设计样式
```

6. 并集选择器

并集选择器是各个选择器通过逗号连接而成的,任何形式的选择器都可以作为并列式选择器的一部分。如果某些选择器定义的样式完全或部分相同,就可以使用并列式选择器为它们定义相同的 CSS 样式,例如:

```
/* h1 标签, id 属性为"myspan"的 span 标签, class 属性为"myclass"的标签具有相同的属性 */
h1, span #myspan, .myclass{...}
```



选择器都是可以综合使用的,可以自由组合。

7. 通配符选择器

通配符选择器用 * 号表示,它是所有选择器中作用范围最广的,能匹配页面中的所有 HTML 标签元素。

3.3.2 其他选择器

3.3.1 节所讲的选择器基本上都能满足页面设计中的常规需求,HTML5 App 开发者必须重点掌握。对于一些特殊的设计需求,还可以使用表 3-4 中的选择器。

表 3-4 其他选择器

选 择 器	例 子	描 述
element > element	div > p	选择父元素为 div 标签的 p 标签(p 标签必须是 div 标签的直接子元素)
element + element	div + p	选择紧跟在 div 标签后面的 p 标签(不是内部)
element1 ~ element2	p ~ ul	选择有相同的父元素中位于 p 元素之后的所有 ul 元素
[attribute]	input[name]	选择所有包含 name 属性的 input 标签
[attribute = value]	input[name = "myname"]	选择 name 属性为 "myname" 的 input 标签
[attribute ^ = value]	input[name ^ = "my"]	选择 name 属性以 "my" 开头的 input 标签
[attribute \$ = value]	input[name \$ = "me"]	选择 name 属性以 "me" 结尾的 input 标签
[attribute * = value]	input[name * = "na"]	选择 name 属性包含有 "na" 的 input 标签
:link	a:link	选择所有未被访问的超链接
:visited	a:visited	选择所有已被访问的超链接
:active	a:active	选择所有活动链接
:hover	div:hover	选择鼠标悬停的 div 标签
:focus	input:focus	选择所有获取焦点的 input 标签
:first-letter	p:first-letter	选择 p 段落中的首字母
:first-line	p:first-line	选择 p 段落中的首行
:first-child	p:first-child	选择属于父元素的第一个子元素的 <p> 标签
:last-child	p:last-child	选择属于父元素的最后一个子元素的 <p> 标签
:before	p:before{content: "测试";}	在每个 <p> 标签的内容之前插入文字 "测试"
:after	p:after{content: "测试";}	在每个 <p> 标签的内容之后插入文字 "测试"
:first-of-type	div p:first-of-type	选择 div 标签里面的第一个 p 标签
:last-of-type	div p:last-of-type	选择 div 标签里面的最后一个 p 标签
:nth-child(n)	li:nth-child(2)	选择属于其父元素的第 2 个 li 标签
:nth-last-child(n)	li:nth-last-child(2)	选择属于其父元素的倒数第 2 个 li 标签
:empty	div:empty	选择没有子元素的 div 标签
:not	li:not(:last-child)	选择除去最后一个 li 元素的其他所有 li 标签

3.4 尺寸属性

为了控制各标签显示的大小,CSS 提供了一系列的尺寸属性,具体如表 3-5 所示。

表 3-5 尺寸属性

属 性	描 述
width	设置标签元素的高度
height	设置标签元素的高度
max-width	设置标签元素的最大宽度。在内容没有达到 max-width 设定的值时,HTML 标签元素的宽度可以随内容自适应;一旦达到了,则宽度不再变化
min-width	设置标签元素的最小宽度。在内容没有达到 min-width 设定的值时,HTML 标签元素的宽度保持为 min-width 设定值;达到了,则宽度随内容自适应
max-height	设置标签元素的最大高度。在内容没有达到 max-height 设定的值时,HTML 标签元素的高度可以随内容自适应;一旦达到了,则高度不再变化
min-height	设置标签元素的最小高度。在内容没有达到 min-height 设定的值时,HTML 标签元素的高度保持为 min-height 设定值;达到了,则高度随内容自适应

3.5 文本样式属性

1. 普通文本样式

为了方便地控制页面中文字的各种属性,CSS 提供了一系列的样式属性,如表 3-6 所示。

表 3-6 文本样式属性

方 式	描 述
color	设置文字颜色
font-size	设置文字大小
font-weight	设置字体的粗细,默认 normal 标准体,bold 粗体,bolder 更粗,lighter 更细,100~900 整数(100 倍整数倍),400 等于 normal,700 等于 bold
font-family	设置字体,可以同时指定多个,以逗号隔开,如果不支持第一个字体,则会尝试下一个,以此类推,若指定的字体没有安装时,会使用浏览器默认的字体系。英文字体不需要加引号,中文字体需要,英文字体应位于中文字体之前,例如:body{font-family: Arial,“微软雅黑”}
font-style	设置为 italic 时使用斜体
letter-spacing	设置字符之间距离
word-spacing	设置英文单词之间距离
line-height	设置行间距
text-transform	capitalize 是首字母大写,uppercase 全部大写,lowercase 全部小写
text-decoration	underline 是下划线,overline 上划线,line-through 删除线
text-align	设置水平对齐方式,left 是左对齐,right 是右对齐,center 是居中
text-indent	设置首行缩进处理
text-shadow	为页面中的文本添加阴影效果
text-overflow	设置文本溢出时的处理,clip 为修剪,ellipsis 为用省略号“...”标示修剪文本

下面用一个例子示范 CSS 的尺寸属性、文本样式属性的使用。

【例 3-1】 CSS 尺寸属性、文本样式属性应用示范,代码如下:


```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="initial-scale=1.0, maximum-scale=1.0, user-scalable=no"/>
    <title>CSS 尺寸属性、文本样式属性应用示范</title>
    <style>
      div,p{
        font-size: 14px;
        font-family: "微软雅黑";
      }
      .mypara{
        color: #222222;
        background-color: #EC971F;
        min-height: 20px;
        text-indent: 2em;
      }
      #parent p:first-of-type{
        font-weight: bold;
        font-style: italic;
        text-align: center;
      }
    </style>
  </head>
  <body>
    <div id="parent" style="background-color: red;">
      <p class="mypara">
        笑傲江湖
      </p>
      <p class="mypara">
        田伯光将刀刀架在他喉头,喝道: "还打不打?打一次便在你身上砍几刀,纵然不杀你,
        也要你肢体不全,流干了血."令狐冲笑道: "自然再打!就算令狐冲斗你不过,难道我风太师叔袖手
        不理,任你横行?"田伯光道: "他是前辈高人,不会跟我动手."说着收起单刀,心下毕竟也甚惴惴,生
        怕将令狐冲砍伤了,风清扬一怒出手,看来这人虽然老得很了,糟却半点不糟,神气内敛,眸子中英
        华隐隐,显然内功着实了得,剑术之高,那也不用说了,他也不必挥剑杀人,只须将自己逐下华山,那
        便糟糕之极了."
      </p>
    </div>
  </body>
</html>

```

在 Chrome 中浏览该页面,效果如图 3-3 所示。

例 3-1 示范了:

- 内联样式和内嵌样式的使用;
- 如何定义各种 CSS 选择器,包括 id 选择器、标签选择器、类选择器、并列选择器、后

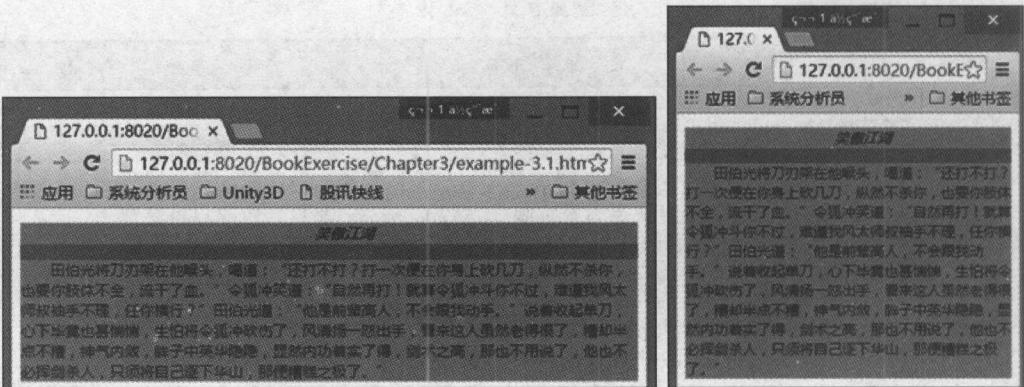


图 3-3 CSS 尺寸和文本样式示例

代选择器,还有 CSS3 比较有特色的第一个子标签选择器;

- 文本字体、颜色、粗细、斜体、居中的使用,中文段落首行缩进两格的使用;
- 尺寸中固定高度和宽度的设定,缩放浏览器窗口,可以看出 min-height 的使用。



在 HBuilder 中书写 CSS 属性时,根据输入的字符, HBuilder 会有相应的智能提示,并对该属性作相应的解释,显示它在各浏览器中的兼容性,如图 3-4 所示。

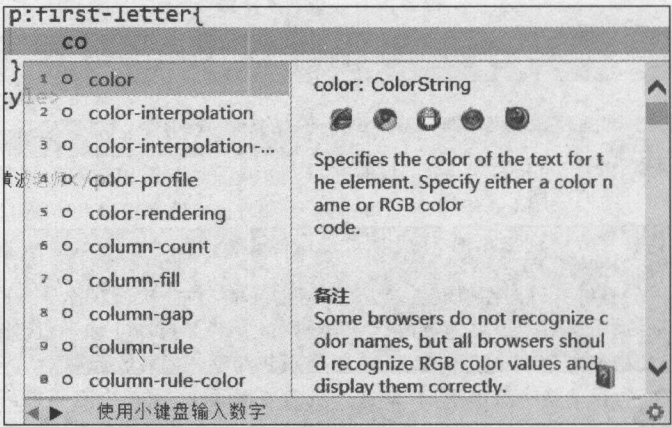


图 3-4 HBuilder 中 CSS 的智能提示

2. 网络字体

在 CSS3 中提供了一个@font-face 规则,用来定义服务器字体,开发者可以使用任何喜欢的字体,而不用考虑客户端是否安装了相应字体。当用户在浏览该页面时,字体会自动下载到客户端(App 开发是将字体文件打包进安装包)。

@font-face 的基本语法格式为:

```
@font-face {
    font-family: "自定义字体名称";
```

```
src: url("字体路径");
}
```

下面通过一个“毛泽东字体”的案例,来演示@font-face的具体用法。

【例 3-2】 网络字体示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <title>网络字体示范</title>
    <link rel = "stylesheet" href = "css/example-3.2.css" />
  </head>
  <body>
    <p id = "mypara">
      北国风光,千里冰封,万里雪飘。
    </p>
  </body>
</html>
```

其中,链接的 CSS 文件 example-3.2.css 的内容如下:

```
html{
  font-size: 28px;
}
@font-face {
  font-family:mzd;
  src: url("../fonts/mzd.ttf");
}
#mypara{
  font-family: mzd;
  font-size: 0.9rem;
}
```

页面在 Chrome 中浏览,效果如图 3-5 所示,CSS 里的 rem 单位是相对于 html 选择器中定义的字体大小,段落中的文字会随着 html 选择器中字体的大小改变。

北国风光,千里冰封,万里雪飘。

图 3-5 网络字体显示效果



有字库(<http://www.youziku.com/>)提供了中文网络字体的在线服务(不适用于 App 开发)。

3.6 CSS 高级特性

3.6.1 继承性

所谓继承性是指书写 CSS 样式时,子标签会自动继承父标签的某些样式,例如文本的颜色和字号。利用这个特性,在设计页面时,不必在标签元素的每个子元素上再重复书写样式了,例如下面这段代码中,只是为父级元素设计了字体颜色,但它的子级元素会自动继承下来。

```
<div id="parent" style="color: red;">
    父及元素中的文字
    <div id="child">
        子级元素中的文字
    </div>
</div>
```

恰当地使用继承这个特性可以简化代码,降低 CSS 样式的复杂性。例如字体、字号、颜色、行距就可以在 body 选择器中统一设置,然后通过继承影响页面中的所有文本。但是并非所有的 CSS 属性都可以被继承,例如下面的这些 CSS 属性就不具备继承性:

- 边框属性;
- 边距和填充属性;
- 背景属性;
- 定位属性;
- 尺寸属性。

3.6.2 CSS 层叠性和优先级

所谓层叠性是指对于同一个标签元素是可以设计多个 CSS 样式的,而 HTML 标签在页面上的最终显示效果是多种 CSS 样式的叠加结果,例如下面的设计,在页面上有这样一段 HTML 代码:

```
<p style="color: red;" id="mypara" class="special">段落文本</p>
```

在这个 p 标签 style 属性中设计文本颜色为红色,又使用内嵌样式为它设计了如下样式:

```
<style>
    #mypara{
        font-size: 14px;
    }
    .special{
        font-family: "微软雅黑";
    }
</style>
```

```
}  
</style>
```

最终在页面中浏览后,该段落中的文本样式是几种设计最终叠加的效果,颜色为红色,字体是“微软雅黑”,字体大小为 14px。

这里会存在问题:如果定义 CSS 样式时,出现多个相同的 CSS 属性,而不同的值应用在同一 HTML 标签元素上,它的最终效果又如何决定呢?这里需要对 CSS 样式的优先级进行讲解。

CSS 样式的优先级分为 4 个等级,每个等级代表一种选择器,如表 3-7 所示。

表 3-7 CSS 样式 4 个等级权重

选择器	权重
标签选择器	1
类选择器	10
id 选择器	100
style 属性	1000

对于由多个基础选择器构成的复合选择器(并集选择器除外),其权重为这些基础选择器权重的叠加,例如下面这些 CSS 代码:

```
p span{...} /* 权重为 1+1=2 */  
P.blue{...} /* 权重为 1+10=11 */  
.blue div{...} /* 权重为 10+1=11 */  
p.parent span{...} /* 权重为 1+10+1=12 */  
p.parent .child{...} /* 权重为 1+10+10=21 */  
#header span{...} /* 权重为 100+1=101 */  
#header span.blue{...} /* 权重为 100+1+10=111 */
```

当 CSS 样式叠加时,页面将应用权重最高的样式,另外要注意一些特殊情况:

- 继承样式的权重为 0,也就是说子标签元素的样式会覆盖继承来的样式。
- 内联样式优先,也就是标签的 style 属性定义的样式,因为它的权重很高。
- 权重相同时,CSS 遵循的是就近原则,也就是说,后应用的样式优先级更大。
- CSS 定义了一个 !important 语法,它的作用是赋予最大的优先级,也就是说,不管权重如何以及样式位置的远近,!important 都有最大的优先级,例如下面这个样式:

```
#mydiv{color:red!important;} /* 不管其他样式如何设置,最终一定是红色文字 */
```

3.6.3 Chrome 调试 CSS

下面使用 Chrome 浏览页面,利用它的页面调试工具,对页面 HTML 和 CSS 进行调

试,进一步掌握 CSS 的使用。

1. 页面 CSS 样式查看及调试

在 Chrome 中浏览例 3-1 的页面,把鼠标移到第二段文字上面后,单击鼠标右键,在弹出的菜单中选择“检查”(或按 Ctrl+Shift+I 键),如图 3-6 所示,打开 Chrome 的“开发者工具”,如图 3-7 所示。

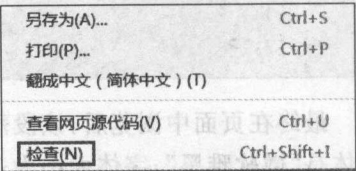


图 3-6 “检查”菜单

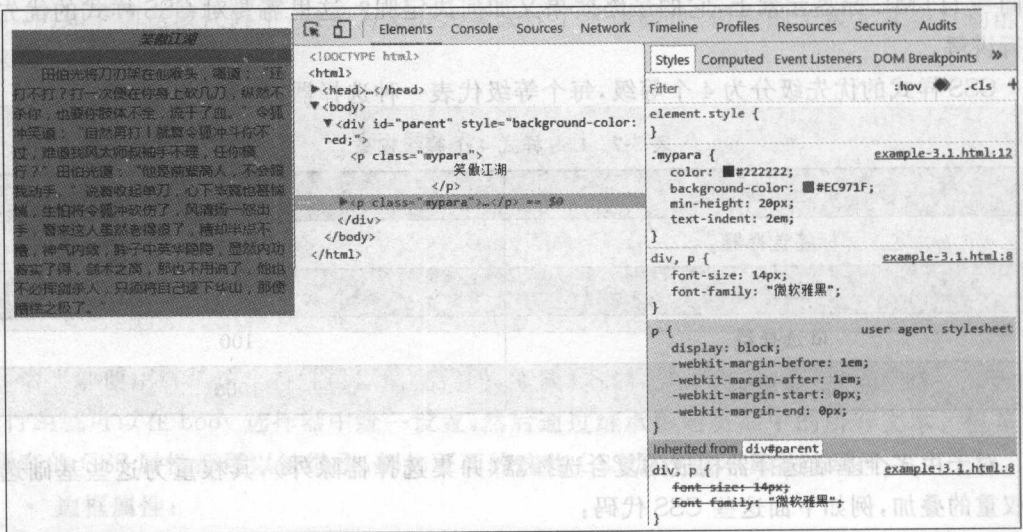


图 3-7 Chrome“开发者工具”界面

在这个工具中可以看到,界面的 Elements 选项卡中显示出了页面的 HTML 源代码,并且鼠标选择“检查”的位置对应的 HTML 标签代码背景会以灰色显示,在图 3-7 中的右侧,从下到上显示了第二个<p>标签的 CSS 样式层叠过程,CSS 样式显示有删除线,如图 3-8 所示,表示有同样的样式设置进行了覆盖。如果 CSS 样式书写是有问题的,界面上也会有明显的提示,如图 3-9 所示。

使用这样的工具,对于标签元素最终叠加出现的效果的过程一目了然,出现了问题也便于分析(例如选择器权重不够,设计的 CSS 样式应用不上或样式书写错误)。单击如图 3-10 所示的“即时 inspect 按钮”高亮后,鼠标在 HTML 页面上划过不同的 HTML 标签时,Elements 选项卡中的 HTML 代码会自动跟随切换,单击鼠标时,和前面的 CSS 观察效果一样,自动显示样式的层叠过程。

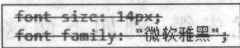


图 3-8 CSS 样式被覆盖

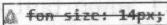


图 3-9 CSS 样式不能被识别



图 3-10 即时 inspect 按钮



在实际的开发中,经常需要不断修改 CSS 属性值,以达到最佳的视觉效果。

Chrome 提供了“即改即所得”的修改功能,如图 3-11 所示,鼠标放在某段样式设计上时,它内部设计的 CSS 属性前面都会出现复选框,单击鼠标左键,进入编辑状态,可以直接

添加新的 CSS 属性,页面立刻会自动应用新的 CSS 样式。

也可以试着取消选中某个 CSS 样式,如图 3-12 所示,取消 text-indent 属性后,该属性被打上了删除线,页面中段落首行空两格的效果也自动消失了。

还可以用鼠标单击某个 CSS 属性的值,使其可修改,如图 3-13 所示,页面会根据修改的值立即作出响应。

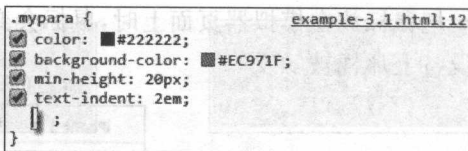


图 3-11 样式可编辑效果

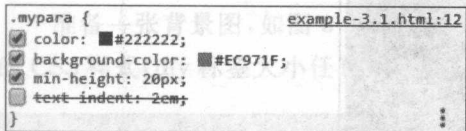


图 3-12 取消某个样式效果

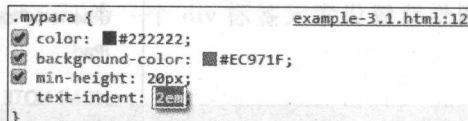


图 3-13 修改某个样式效果



“即改即所得”调试效果中对 CSS 的设置和修改不会自动保存,可以记下对应的文件和行号,调试好之后在相应文件及位置进行修改再保存。

2. 模拟不同的移动设备查看页面

之所以在 HTML5 App 开发中推荐使用 Chrome 浏览器,除了它具有强大的页面调试功能,还有一项功能非常好用,也是目前移动应用开发中经常需要考虑的设备适配问题。由于移动设备的分辨率变化较大,可以使用 Chrome 的模拟功能,进行不同设备分辨率下的查看。具体方法是:打开 Chrome 的“开发者工具”后,单击左上角的“移动设备切换”按钮,如图 3-14 所示,进入移动设备查看页面。效果如图 3-15 所示。

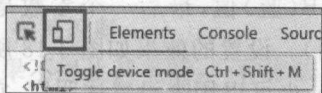


图 3-14 移动设备切换按钮

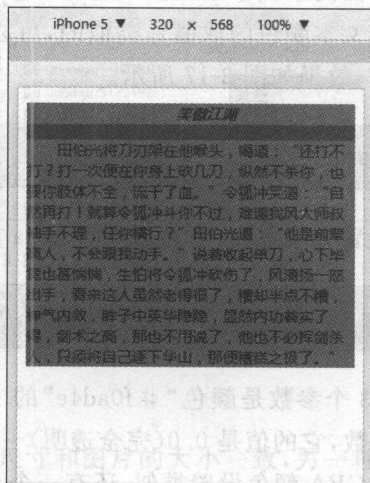


图 3-15 移动设备查看页面

Chrome 已经内置了一些移动设备的分辨率,例如 iPhone 5、iPhone 6、iPhone 6 plus,如图 3-16 所示,你可以选择不同的设备查看页面效果。如果没有相应的设备或分辨率,还可以单击 Edit 选项进行添加。

把鼠标放在模拟器页面上时，鼠标会变成一个指尖大小的圆圈，同时触摸事件会像在手机设备上那样被触发。



图 3-16 移动设备切换菜单

3.7 背景属性

页面能通过背景图像给人留下深刻的印象，所以合理控制背景颜色和图像至关重要。本节将介绍 CSS 控制背景的一些样式属性。

3.7.1 设置背景颜色

在前面的示例中已经使用过 background-color 属性来设置 HTML 标签元素的背景色，其颜色和文本颜色的取值是类似的，可以使用表 3-3 中的所有设置。例如采用十六进制的设置方式，效果如图 3-17 所示。

```
background-color: #f0ad4e;
```

在设置背景色时，还可以采用 rgba 方式来控制背景的透明度，例如：

```
background-color: rgba(240,173,78,0.5);
```

其中，前 3 个参数是颜色“#f0ad4e”的 R(红)、G(绿)、B(蓝)值，最后一个参数是透明度 Alpha 参数，它的值是 0.0(完全透明)~1.0(完全不透明)。设置的效果如图 3-18 所示。

和 RGBA 颜色设置类似，还有一个 CSS 属性 opacity 也可以用来设置透明度，它的属性值也是介于 0~1 的浮点数，和 RGBA 不同的是，它可以使任何元素呈现透明效果，它的语法如下，效果如图 3-19 所示。

```
opacity: 0.5;
```

2017年1月1日, HBuilder的
开发者数量定格在70万!
HBuilder确认是中国的主流
HTML5开发工具无疑了。

图 3-17 十六进制颜色设置

2017年1月1日, HBuilder的
开发者数量定格在70万!
HBuilder确认是中国的主流
HTML5开发工具无疑了。

图 3-18 RGB 颜色设置

2017年1月1日, HBuilder的
开发者数量定格在70万!
HBuilder确认是中国的主流
HTML5开发工具无疑了。

图 3-19 opacity 透明度设置

3.7.2 设置背景图片

1. 简单设置

准备一张背景图,如图 3-20 所示,使用 background 为一个 div 标签元素设置背景图像
的 CSS 样式(div 标签大小任意)。

```
background: url(imgs/bk.jpg);
```



url() 中代表的是图片所在的路径,可以使用相对路径(是当前 CSS
样式相对图片的路径,不是页面相对图片的路径),也可以使用网络路径。

在 HBuilder 中,输入 url 后,会有路径的自动提示和图片预览,非常方便,如图 3-21 所
示。设置背景图片时,是可以使用多张图片的,例如下面的语法:

```
background: url(img_flwr.gif), url(paper.gif);
```

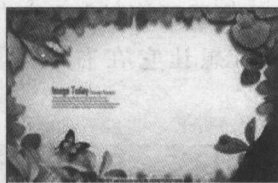


图 3-20 背景图像素材

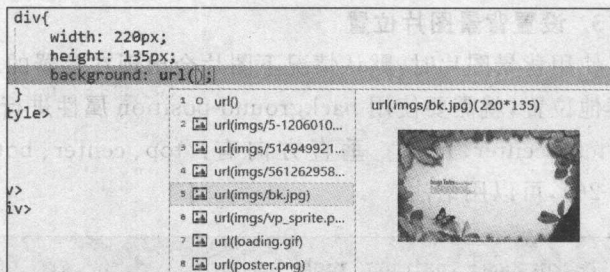


图 3-21 HBuilder 图像路径提示

在 Chrome 中浏览后,会发现背景图片自动沿水平和垂直两个方向平铺,充满整个 div,
效果如图 3-22 所示。

2. 平铺控制

如果不想让其平铺,一种方法是让容器 div 的尺寸和图片的大小一致,另一种方法就是
使用下面的语法进行控制,水平平铺和垂直平铺效果如图 3-23 和图 3-24 所示。

```
background: url(imgs/bk.jpg) no-repeat; //不允许平铺
background: url(imgs/bk.jpg) repeat-x; //水平平铺
background: url(imgs/bk.jpg) repeat-y; //垂直平铺
```

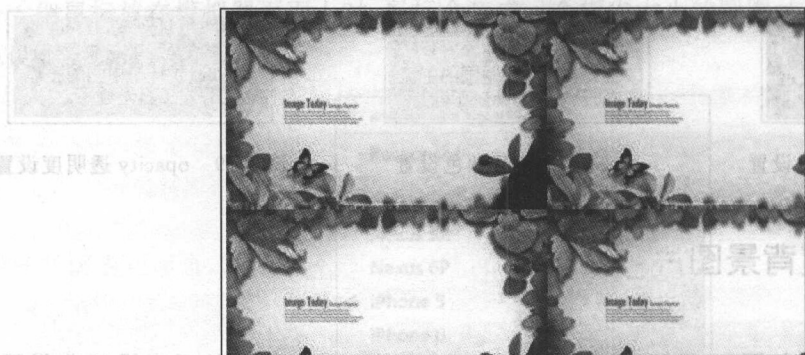



图 3-22 背景图像自动平铺



图 3-23 背景图像水平平铺

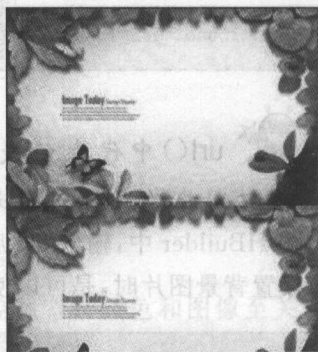


图 3-24 背景图像垂直平铺

3. 设置背景图片位置

使用背景图片时,默认情况下图片会出现在容器的左上角,如图 3-25 所示,若需要出现在其他位置,就需要使用 `background-position` 属性进行控制。它的值用于控制水平方向的有: `left`、`center`、`right`; 垂直方向有: `top`、`center`、`bottom`。例如,想让它右下角(见图 3-26),可以用语法:

```
background-position: right bottom;
```

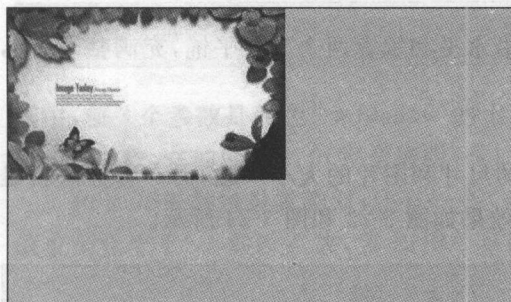


图 3-25 背景图像默认在左上角



图 3-26 背景图像控制在右下角

4. CSS Sprite 使用

CSS Sprite 又称为“CSS 精灵”，是一种网页图片应用处理方式。它允许将一个页面涉及的所有零星图片都包含到一张大图中去，这样一来，当访问该页面时，载入的图片就不会像以前那样一幅一幅地慢慢显示出来了，它可以显著提高图片加载的效率。例如下面的图片素材，如图 3-27 所示，所有的图标都在一张图片上。那如何控制其显示哪一部分图片呢？这得借助 background-position 属性，下面用一个例子来演示 CSS Sprite 技巧。



图 3-27 CSS Sprite 图片

【例 3-3】 CSS Sprite 技巧应用示例，代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title>CSS Sprite 应用技巧</title>
    <style>
      div{
        width: 40px;
        height: 44px;
        background: url(imgs/spritetest.png) no - repeat;
      }
      #div2{
        background - position: - 40px 0;
      }
      #div3{
        background - position: - 80px 0;
      }
    </style>
  </head>
  <body>
    <div id = "div1"></div>
    <div id = "div2"></div>
    <div id = "div3"></div>
  </body>
</html>
```

例 3-3 使用了 3 个 div 分别作为三个图标的容器，容器的大小控制为图标的大小，它们使用的都是同一张图片作为背景，只是显示的图片部分不同。这里需要使用 background-position 进行相应的坐标位置控制。坐标的计算方法是将容器和背景的图片左上角重合，考虑图片如何移动，让其显示在容器中。在 Chrome 中浏览的效果如图 3-28 所示。



图 3-28 CSS Sprite 应用

在 HTML5 App 开发中会涉及背景图片的适配问题,各种移动设备屏幕尺寸不一样,而制作不同大小的图片又不现实,这时候 CSS 属性 `background-size` 就可以发挥重要作用了,它的语法形式为:

```
background-size:属性值 1 属性值 2;
```

属性值	说 明
像素值	设置背景图片的像素高度和宽度,第一个是宽度,第二个是高度。若只设一个值,则第二个值默认为 auto
百分比	以父元素的宽度和高度百分比设置背景图片的高度和宽度。只设一个值,则第二个值默认为 auto
cover	保持图像的纵横比并将图像缩放成将完全覆盖背景定位区域的最小大小
contain	保持图像的纵横比并将图像缩放成将适合背景定位区域的最大大小

下面通过一个简单的例子来进行演示。

【例 3-4】 background-size 实现背景图片的适配,代码如下:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF - 8">
        <title>background - size 实现背景图片的适配</title>
        <meta name = "viewport" content = "width = device - width,
initial - scale = 1,maximum - scale = 1,user - scalable = no">
    <style>
        html,body{
            margin: 0;
            padding: 0;
            width: 100%;
            height: 100%;
        }
        #mydiv{
            width: 100%;
            height: 100%;
            background: url(imgs/mobilebk.jpg) no - repeat;
            background - size: 100% 100%;
        }
    </style>

```



```

</style>
</head>
<body>
  <div id="mydiv">
    </div>
  </body>
</html>

```

在 Chrome 中浏览后,打开它的“开发者工具”,切换到移动设备模式,切换不同的设备,可以看到它的效果,在不同的手机分辨率下都实现了背景的自适应,如图 3-29 所示。

如果把“background-size: 100%”修改成“background-size: contain”,则有可能出现空白区域这种情况(因为要保持纵横比),如图 3-30 所示。cover 正好相反,如果图片的比例和容器相差很大,某些部分可能会截取掉,不会显示。

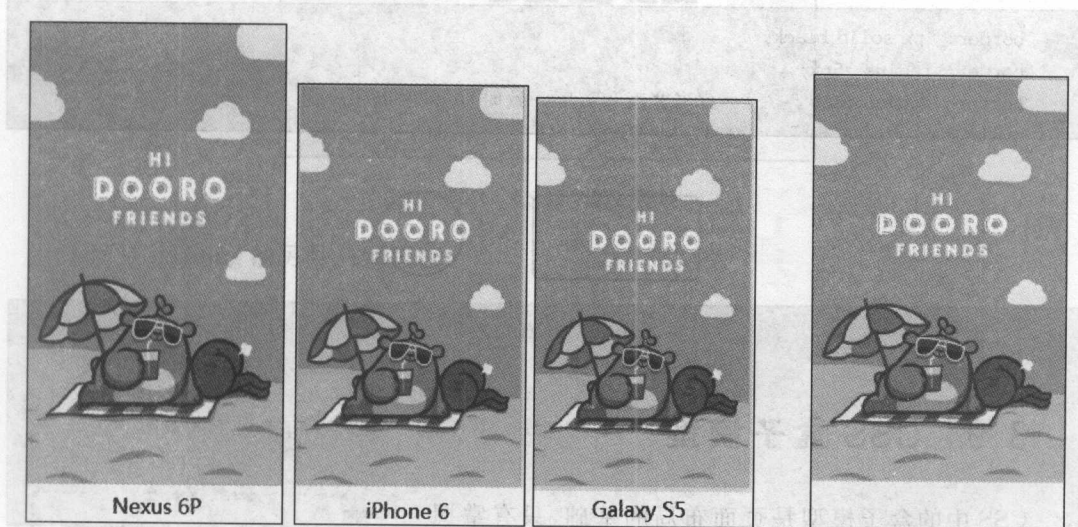


图 3-29 background-size 的适配性

图 3-30 contain 的空白情况

3.8 边框属性

使用 CSS 边框属性可以创建出效果出色的边框,并且可以应用于任何元素。它的语法形式为:

```
border: 1px solid black;
```

其中,1px 代表边框粗细,solid 代表线,图 3-31 中还有其他设置的示例,black 表示线条颜色为黑色。

对应于四条边框,还可以使用 border-top、border-bottom、border-left、border-right 进行分别控制。

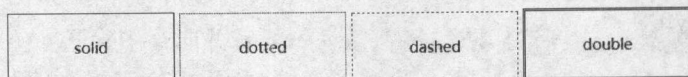


图 3-31 不同的边框



我们一般使用 `border:none;` 来消除边框。

在页面设计中,经常需要设置圆角边框,可以使用 `border-radius` 完成圆角化效果,它的语法形式为:

`border-radius`: 左上角圆角半径 右上角圆角半径 右下角圆角半径 左下角圆角半径

如果 4 个值相同,可以只用一个参数,例如下面的代码,其显示效果如图 3-32 所示。

```
border: 2px solid black;
border-radius: 5px;
border-radius: 50%; //当宽度和高度一致时,可以生成一个圆
```



图 3-32 圆角效果

3.9 CSS 盒子模型

CSS 中的盒子模型是页面布局的基础,只有掌握了盒子模型和各种规律和特征,才能更好地控制页面中各元素呈现效果。盒子模型如图 3-33 所示,就是把 HTML 标签元素看作是一个矩形的盒子或容器。每个矩形都由元素的内容、内填充(padding)、边框(border)和外边距(margin)组成。

- 盒子的总宽度=内容宽度+左右填充+左右外边距+左右边框宽度
- 盒子的总高度=内容高度+上下填充+上下外边距+上下边框宽度

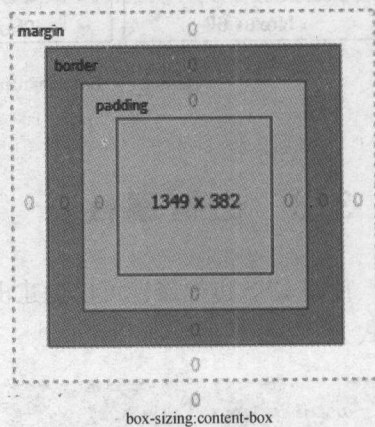


图 3-33 CSS 盒子模型示意

3.9.1 内填充属性

内填充属性指的是 HTML 标签元素内容与边框之间的距离。使用 CSS 属性 `padding` 进行设置的代码形式为:

padding: 上填充距离 右填充距离 下填充距离 左填充距离

它的参数有 4 个,除了第 1 个参数,其他 3 个是可选参数,例如,对于 img 标签使用 padding 属性进行设置如下,效果如图 3-34 所示。

```
padding:25px 50px 75px 100px;  
//上、右、下、左填充距离分别为 25px 50px 75px 100px
```



图 3-34 CSS 内填充效果

下面是 padding 的另外一些用法:

```
padding:25px 50px 75px;  
//上填充距离为 25px、左右填充距离为 50px、下填充距离为 75px  
padding:25px 50px;  
//上下填充距离为 25px、左右填充距离为 50px  
padding:25px;  
//上下左右填充距离均为 25px
```

也可使用 padding-top、padding-bottom、padding-left、padding-right 这 4 个属性分别进行单独设置。

3.9.2 外边距属性

从盒子模型的定义可以看出,页面上的内容是由很多盒子排列而成的,要想拉开盒子与盒子之间的距离,合理布局页面,就需要为盒子设置外边距。CSS 中使用的是 margin 属性,它的语法和 padding 类似,参数也是 1~4 个。

margin: 上外边距 右外边距 下外边距 左外边距

同样可以用 margin-top、margin-bottom、margin-left、margin-right 这 4 个属性分别进行单独设置。

在水平方向,两个盒子之间的距离是左边盒子的右边距与右边盒子的左边距之和,如图 3-35 所示,两个 img 标签元素之间的水平外边距为 20px。margin 和 padding 的值还


可以设置为负数,如图 3-36 所示,设为负值后,两个标签元素会出现重叠效果。



图 3-35 水平边距相加



图 3-36 margin 为负值为叠加

但是在垂直方向就不一样了,当两个垂直相邻的块级元素的上下两个边距相遇时,外边距会产生重叠现象,且重叠后的外边距等于其中较大者。如图 3-37 所示。

在 Chrome 中,当打开“开发者工具”查看某个标签元素时,它的边距和填充会自动显示在工具的窗口中,如图 3-38 所示。

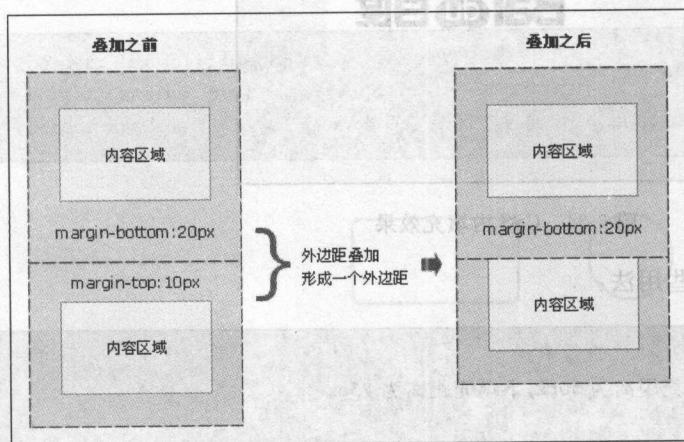


图 3-37 上下边距叠加问题

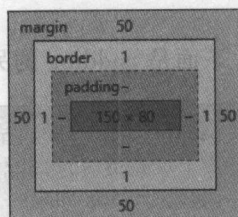


图 3-38 Chrome 边距查看

页面设计时经常使用下面的代码来实现内容水平居中,0 表示上下外边距为 0,auto 表示左右则根据宽度自适应相同值。

```
margin: 0 auto;
```



由于浏览器自带的样式有自带的内填充和外边距,所以在 HTML5 App 开发中常用 `html,body{margin:0;padding:0;}` 去除边距和填充。

3.9.3 box-sizing 属性

当一个盒子的总宽度确定之后,要想再添加边框或内填充,往往需要重新计算 width 属性值,这样才能保证盒子宽度不变,但修改非常麻烦。使用 box-sizing 属性可以解决这个问题,它用于定义盒子的宽度值和高度值是否包含元素的内填充和边框,语法格式如下:

```
box-sizing: content-box|border-box;
```

content-box 表示内填充和边框不包括在宽度和高度之内,而 border-box 则表示内填充和边框包括在宽度和高度之内。

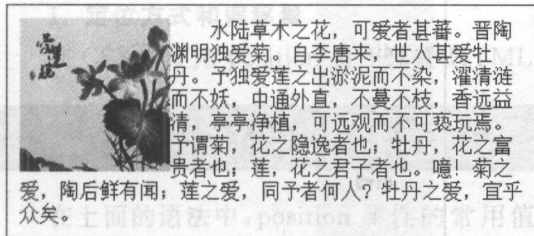
3.10 浮动和定位

默认情况下,页面中的 HTML 元素会按从上到下或从左到右的顺序将一个个盒子罗列出来,如果按这种方式对页面进行排版,页面会非常单调和混乱。为了让页面版面更丰富合理,在 CSS 中可以对 HTML 标签元素实现浮动和定位。

3.10.1 浮动

1. 设置浮动

在页面开发中灵活地设置浮动,常常可以得到意想不到的效果,例如文字环绕功能(如下列左图所示)和流行的“瀑布流”列表(如下列右图所示)等。



文字环绕功能



“瀑布流”列表

作为 CSS 重要属性,浮动属性在页面布局中至关重要。可以通过 float 属性来设置浮动。所谓浮动是指浮动的 HTML 标签元素会脱离标准文档流的控制,移动到父元素中指定位置的过程。其语法形式为:

```
float:left|right;
```

下面通过一个例子来学习 float 的用法。

【例 3-5】 浮动的应用示例。

(1) 打开资源包中提供的练习文件“example-3.5-prepare.html”。

(2) 使用 Chrome 浏览该页面,效果如图 3-39 所示。

(3) 在 style 标签中,添加新的样式,如下:

```
#parent #div1{float: right;}
```

(4) 刷新页面后,可以看到如图 3-40 所示的效果,当把 div1 向右浮动时,它脱离文档流并且向右移动,直到它的右边缘碰到包含框的右边缘,而原有的 div2 和 div3 自动向上占位。

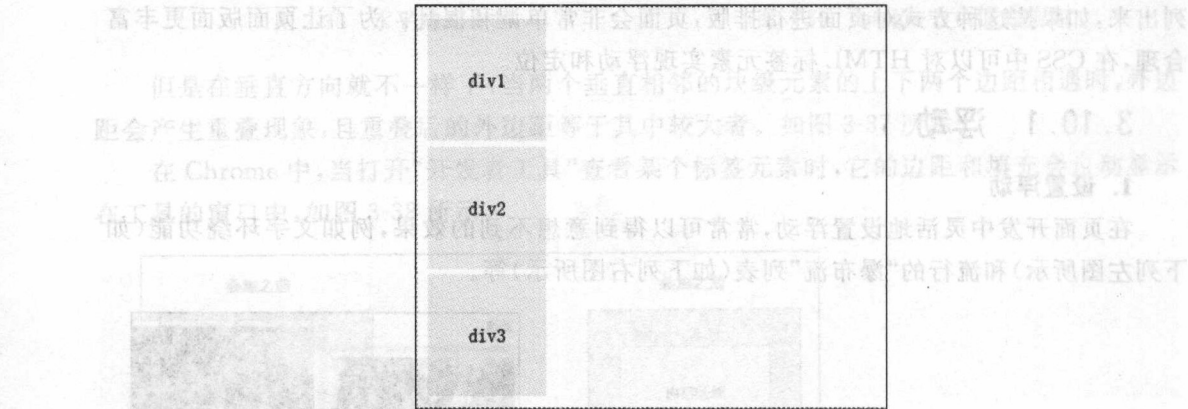


图 3-39 初始效果

(5) 将在(3)中添加的样式修改为左浮动 float:left,它脱离文档流并且向左移动,直到它的左边缘碰到包含框的左边缘。因为它不再处于文档流中,所以它不占据空间,实际上覆盖住了 div2,使 div2 从视图中消失,如图 3-41 所示。

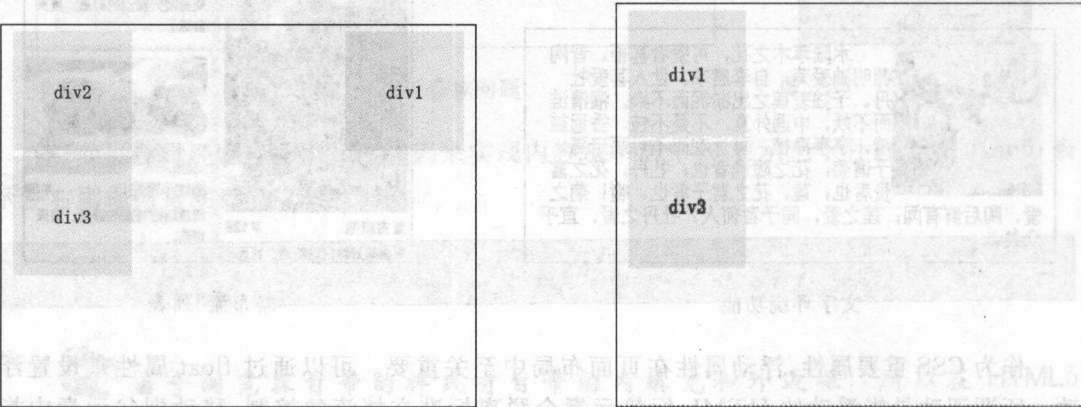


图 3-40 div1 右浮动效果

图 3-41 div1 左浮动效果

(6) 将(5)中的浮动代码注释去掉,使用/* float: left; */。

(7) 修改#parent div{...}中的 width 属性值为 150px,并添加属性“float: left;”,让 div1、div2、div3 都向左浮动,刷新页面后,容器无法容纳水平排列的三个浮动元素,那么其他浮动块向下移动,直到有足够的空间,如图 3-42 所示。

(8) 修改(3)中的样式,添加高度属性“height:200px;”,刷新后的效果见图 3-43。浮动元素的高度不同,那么当它们向下移动时,可能被其他浮动元素“卡住”(即形成瀑布流)。

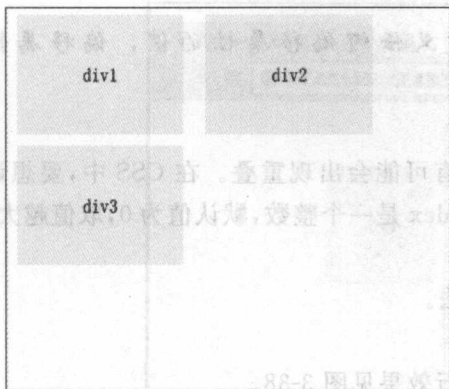


图 3-42 容器宽度不够

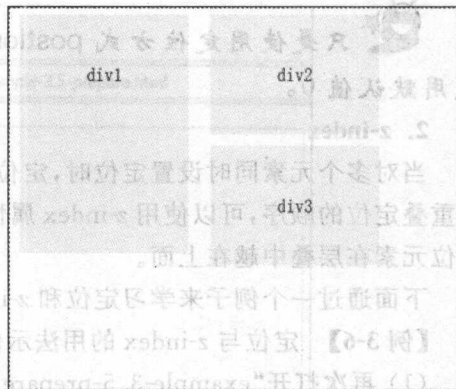


图 3-43 浮动中被卡住

2. 清除浮动

CSS 中的 `clear` 属性定义了 HTML 元素的哪边(左边或右边或两边)不允许出现浮动元素。它的设置语法形式为:

```
clear: left|right|both;
```

3.10.2 定位

浮动布局虽然灵活,但却无法对 HTML 标签元素实现精确控制。在 CSS 中,还可以使用定位属性来进行精确定位。下面对它定位属性的几种控制方式作详细讲解。

1. 定位方式和偏移量

在 CSS 中使用 `position` 属性设置 HTML 标签元素的定位方式。它的语法格式如下:

```
position: static|relative|absolute|fixed;
```

在上面的语法中, `position` 属性的常用值有 4 个, 分别表示不同的定位模式, 具体如表 3-9 所示。

表 3-9 position 属性常用值

属性值	说 明
static	静态定位,即各 HTML 标签元素在 HTML 文档流中默认位置
relative	相对定位,相对于 HTML 标签元素本来该在 HTML 文档流中显示的位置
absolute	绝对定位,依据最近已经实现过定位(相对、绝对、固定)的父元素进行定位,若所有父元素都没有定位,则根据 body 元素(浏览器窗口)定位
fixed	固定定位,以浏览器窗口作为参照物。不管浏览器滚动条如何滚动或浏览器窗口大小如何变化,始终显示在固定的位置

定位模式(`position`)仅用于定义 HTML 标签元素以哪种方式定位,并不能确定它的具体位置,可以通过偏移属性 `top`、`bottom`、`left`、`bottom` 的组合来精确定义元素的位置。



只是使用定位方式 `position`，没有定义任何偏移属性的值，偏移属性使用默认值 0。

2. z-index

当对多个元素同时设置定位时，定位元素之间有可能会出现重叠。在 CSS 中，要想调整重叠定位的顺序，可以使用 `z-index` 属性控制，`z-index` 是一个整数，默认值为 0，取值越大，定位元素在层叠中越在上面。

下面通过一个例子来学习定位和 `z-index` 的用法。

【例 3-6】 定位与 `z-index` 的用法示例。

(1) 再次打开“example-3.5-prepare.html”，运行效果见图 3-38。

(2) 在 `style` 标签中，为 `div1` 添加相对定位代码，如下：

```
#parent #div1{  
    position: relative;  
    left:150px;  
    top:50px;  
}
```

(3) 在 Chrome 中浏览后，效果如图 3-44 所示，这里会发现，对 HTML 标签元素应用相对定位后，即使它不在原位置了，但它本身所占的位置会保留，不允许其他元素使用。

(4) 将 `div1` 的定位方式修改成绝对定位“`position: absolute;`”，在 Chrome 中浏览后，效果如图 3-45 所示。由于它的父级元素中没有实现过定位，所以 `div1` 的偏移是相对浏览器的，它原有的位置不再保留，其他元素会自动占用。

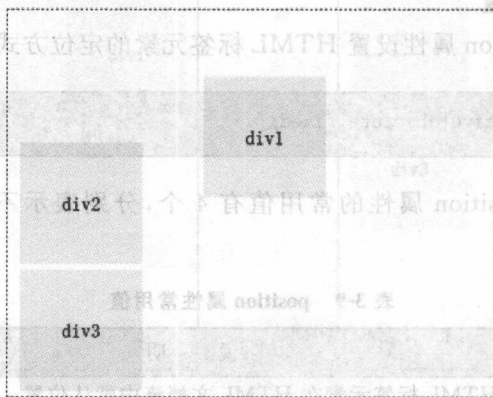


图 3-44 `div1` 的相对定位

(5) 在 `style` 标签中，对 `parent` 这个 `div` 进行定位设置，使用相对或绝对定位都可以，例如，这里使用“`position: relative;`”，这时刷新页面，`div1` 的位置偏移会自动切换到相对 `parent`，如图 3-46 所示。

(6) 在选择器 `#parent div` 中添加绝对定位属性“`position: absolute;`”，刷新页面后的效果如图 3-47 所示。

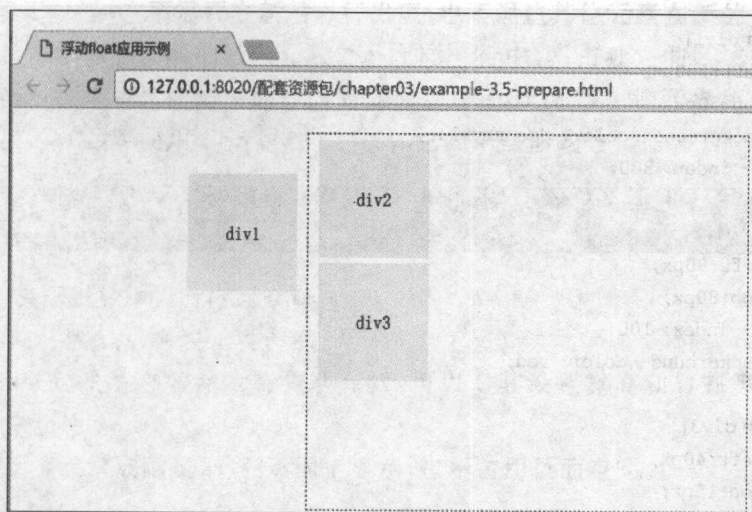


图 3-45 parent 未实现过定位

(7) 修改 style 标签中的样式属性,代码如下,刷新页面后,效果如图 3-48 所示。3 个 div 在 z-index 的控制之下出现了重叠效果,可以试着改变这些值大小,实现不同的重叠。

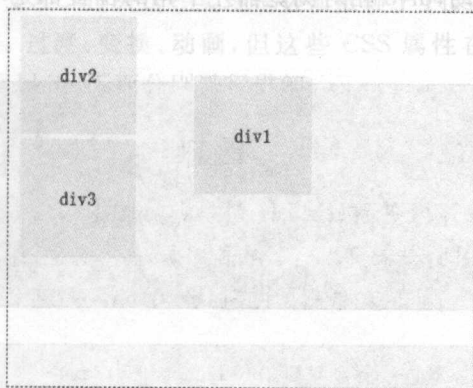


图 3-46 parent 实现了定位

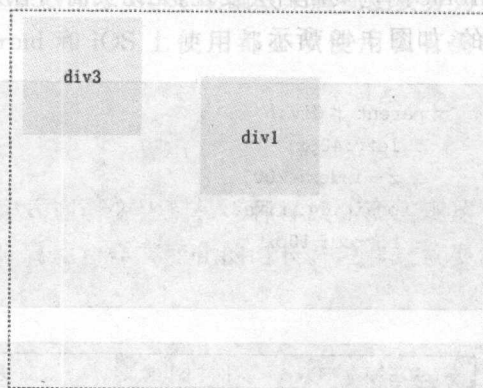


图 3-47 div2 被 div3 覆盖

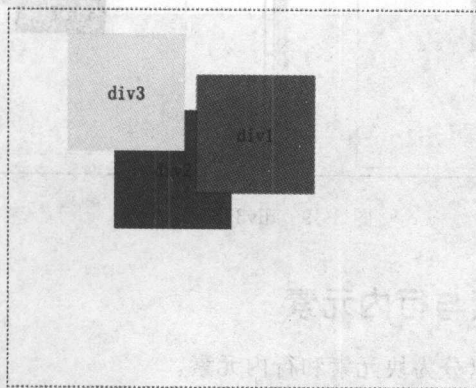


图 3-48 z-index 控制重叠


```

#parent #div1{
    left:150px;
    top:50px;
    background-color: deepskyblue;
    z-index: 300;
}
#parent #div2{
    left:80px;
    top:80px;
    z-index: 100;
    background-color: red;
}
#parent #div3{
    left:40px;
    top:15px;
    z-index: 200;
}

```

(8) 修改 #parent #div3 中的 CSS 属性,代码如下,将 div3 的定位改成了固定定位,Chrome 浏览页面后会发现,无论页面内容如何滚动,div3 相对浏览器左下角的位置都是不变的,如图 3-49 所示。

```

#parent #div3{
    left:40px;
    z-index: 200;
    position: fixed;
    bottom: 10px;
}

```

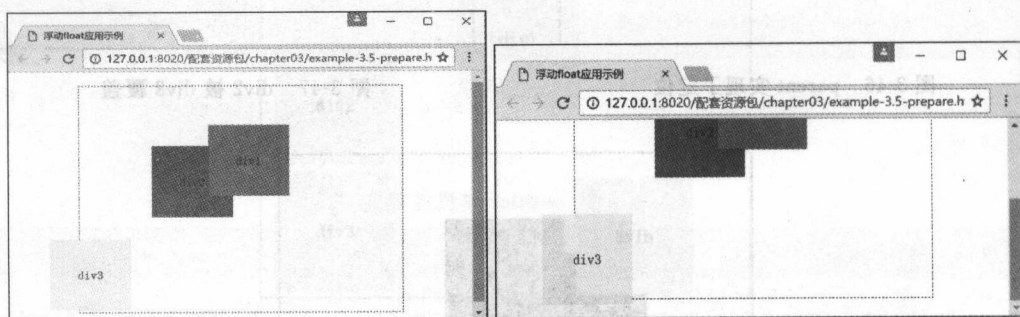


图 3-49 div3 的固定定位

3.10.3 块元素与行内元素

HTML 标签元素可以分为块元素和行内元素。

块元素在页面中以区域块的形式出现,它会独自占据一行或多行,可以对其设置宽度、高度、对齐等属性,常用于页面布局和结构的搭建。常见的块元素标签有 h1~h6、div、p、ul、

li 等。行内元素不一样,它不需要在新的一行出现,也不强迫其他元素在新的一行显示。一个行内元素通常都会和它前后的行内元素显示在同一行中,不占独立的区域。但一般不能设置宽度、高度、对齐等属性。常见的行内元素标签有 span、a 等。块元素和行内元素之间是可以互相转换的。主要使用 CSS 属性 display,它的语法形式为:

```
display: inline|block|inline-block|none;
```

- inline: 将标签显示为行内元素;
- block: 将标签显示为块元素;
- inline-block: 将标签显示为行内块元素,可以对其设置宽高和对齐等属性,但是不会独占一行;
- none: 标签元素被隐藏,不在页面上显示,也不占用页面空间。

3.11 CSS 动画效果

CSS3 的出现,可以使网页上增加不少动画元素,让页面变得更加生动有趣,并且更易于交互。过去这些实现是必须依赖于 Flash 或 JavaScript。CSS3 动画效果属性主要分为三类:过渡、变换、动画,但这些 CSS 属性在 Android 和 iOS 上使用都必须使用私有前缀 -webkit-,下面分别进行讲解。

3.11.1 过渡

CSS 中的过渡是指 HTML 标签元素从一种样式逐渐变化到另一种样式。要实现这个效果,必须考虑两方面内容:变换样式使用的 CSS 属性和样式变化的时长。设置过渡要用 CSS 属性 transition,它的语法形式如下:

```
-webkit-transition: 属性名称 过渡时间 速度曲线 过渡延迟时间
```

其中,前两个参数是必选的,后两个参数是可选的,例如下面这段 CSS 代码:

```
div{
    width: 100px;
    height: 100px;
    background-color: yellow;
    /* 设置宽度和高度变化时间是 2 秒 */
    -webkit-transition: width 2s,height 2s;
}
div:hover{
    width: 200px;
    height: 200px;
}
```

上面这段代码为一个 div 设计了高度和宽度变化的过渡,div:hover 决定了当鼠标悬停

在 div 上时,它的宽度和高度自动增加一倍,但这个变化过程是在 2s 之内缓慢完成的,实现了动画的过程。

如果所有属性变换的时间一样长,过渡属性还可以使用:

```
-webkit-transition: all 2s;
```

至于速度曲线,它的取值及说明见表 3-10 所示。

表 3-10 速度曲线取值及说明

属性值	说 明
linear	规定以相同速度开始至结束的过渡效果,匀速
ease	慢速开始,然后变快,然后慢速结束的过渡效果
ease-in	慢速开始(淡入)的过渡效果
ease-out	慢速结束(淡出)的过渡效果

3.11.2 2D 及 3D 变换

CSS 中的变换属性 transform 可以动态控制 HTML 标签元素,让其在页面中进行移动、缩放、倾斜、旋转,或结合过渡和动画属性产生一些新的动画效果。transform 属性既可以实现 2D 变换,也可以实现 3D 变换。

1. 2D 变换

transform 属性的 2D 变换分为平移、旋转、缩放、倾斜,这些变换操作都是以 HTML 标签元素的中心点为基准进行的,结合过渡可以作出不同的动画效果,下面是它们的语法及示例。

- 平移: 需要使用 translate 方法,指定中心点 X 坐标和 Y 坐标的偏移量,值可以使用负数,表示反方向移动元素,元素平移后,它原有的位置继续保留,语法形式为:

```
-webkit-transform: translate(50px,100px);  
//水平向右偏移 50 像素,竖直向下偏移 100 像素
```

效果如图 3-50 所示(注:虚线都表示 HTML 标签元素的原来状态)。

- 旋转: 需要使用 rotate 方法,角度单位用 deg,默认为顺时针,值可以使用负数,表示逆时针旋转,它的语法形式如下,效果如图 3-51 所示。

```
-webkit-transform: rotate(30deg);  
//绕中心点顺时针旋转 30 度
```

- 缩放: 需要使用 scale 方法,指定宽度和高度的缩放比例,比例可以是小数,语法形式如下,效果如图 3-52 所示。

```
-webkit-transform: scale(0.5,0.5)  
//宽度和高度缩小一半
```


- 倾斜：需要使用 skew 方法，指定相对于 X 轴和 Y 轴的倾斜角度，语法形式如下，效果如图 3-53 所示。

```
-webkit-transform: skew(10deg,10deg);
//相对于 X 轴和 Y 轴都倾斜 10 度
```

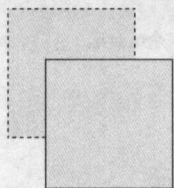


图 3-50 translate 效果

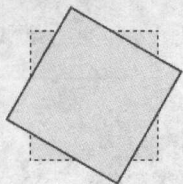


图 3-51 rotate 效果

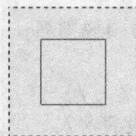


图 3-52 scale 效果

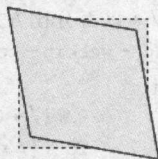


图 3-53 skew 效果

2. 3D 变换

transform 属性的 3D 变换主要是让元素绕 X 轴、Y 轴进行旋转。下面是它们的语法及示例。

- X 轴旋转：需要使用 rotateX 方法，指定旋转角度，单位为 deg，默认为顺时针，值可以使用负数，表示逆时针旋转，语法形式如下，效果如图 3-54 所示。

```
-webkit-transform: rotateX(120deg);
//绕 X 轴顺时针旋转 120 度
```

- Y 轴旋转：使用方法与 rotateX 类似，语法形式如下，效果如图 3-55 所示。

```
-webkit-transform: rotateY(120deg);
//绕 Y 轴顺时针旋转 120 度
```

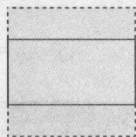


图 3-54 rotateX 效果

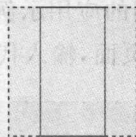


图 3-55 rotateY 效果

3.11.3 动画控制

CSS 除了支持渐变、过渡和变换特效，还可以实现更强大的动画效果，它提供了一个动画控制属性 animation，可以用来设置更复杂的动画效果，例如控制动画次数、逆向动画、动画播放和暂停等。

要使用 animation 动画控制属性，首先要学会定义动画规则，动画规则的定义需要使用 @keyframes 规则，用它来定义动画中的关键帧（表示动画过程中的一个状态），它的语法形式有两种，一种是只设置起始和终止的动画帧，另一种是使用百分比来细化动画帧（百分比

可自由定义),语法形式如下:

```
/* 只设置起始和终止动画帧 */
@-webkit-keyframes 动画规则名
{
    from { /* CSS 属性设置 */ }
    to { /* CSS 属性设置 */ }
}
/* 以百分比方式设定帧 */
@-webkit-keyframes 动画规则名
{
    0% { /* CSS 属性设置 */ }
    25% { /* CSS 属性设置 */ }
    50% { /* CSS 属性设置 */ }
    100% { /* CSS 属性设置 */ }
}
```

定义好动画帧,就可以使用 animation 属性进行动画控制了,它的语法形式如下:

```
-webkit-animation:
    name duration timing-function delay iteration-count direction;
```

- name: 用@keyframes 已定义好的动画规则名。
- duration: 动画花费时间。
- timing-function: 动画速度曲线,取值见表 3-10。
- delay: 动画延迟时间。
- iteration-count: 动画播放次数。
- direction: 动画逆向播放,默认值为 normal,alternate 表示动画轮流反向播放。

下面用一个常见的 CD 播放旋转效果来演示动画控制属性的使用。

【例 3-7】 动画控制的应用示例。

(1) 新建 HTML5 页面,输入代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1,maximum-scale=1,user-scalable=no">
    <title>动画控制属性使用</title>
    <style>
      /* 样式略,请参看本书配套代码 */
    </style>
  </head>
  <body>
    <div id="mScPlrCtn">
```

```

<div id="mScPlr" class="rotateCD"></div>
<div id="mScPlrMask">
  </div>
</div>
</body>
</html>

```

(2) 在 Chrome 中浏览,效果如图 3-56 所示。



图 3-56 初始效果

(3) 在 style 标签中添加如下动画规则和动画控制代码:

```

@-webkit-keyframes CDR{
  from{
    -webkit-transform: rotate(0deg);
  }
  to{
    -webkit-transform: rotate(360deg);
  }
}
.rotateCD{
  -webkit-animation: CDR 3s;
}

```

(4) 刷新页面后会发现: CD 会自动顺时针旋转一圈后停下;

(5) 修改选择器“.rotateCD”样式,加上动画次数,刷新页面后会发现动画自动播放了两次,CD 旋转两圈,代码如下:

```
-webkit-animation: CDR 3s 2;
```

(6) 再次修改,添加逆向播放,刷新页面后发现,CD 顺时针旋转一圈,逆时针旋转一圈,代码如下:

```
-webkit-animation: CDR 3s 2 alternate;
```


(7) 要想保持 CD 一直匀速旋转,需要修改如下:

```
-webkit-animation: CDR 3s infinite linear;
```



动画控制中还有个很有用的属性 `animation-play-state`, 它有两个值: `paused` 和 `running`。它和 JavaScript 结合使用, 可以控制动画的播放和暂停。

3.12 其他一些常用的 CSS 属性

在 HTML5 App 开发中,还有可能会用到一些 CSS 属性,这里简单介绍如下。

- `overflow`: 当容器的内容超过容器自身的大小时,内容溢出的显示方式,其语法格式如下:

```
overflow: visible|hidden|auto|scroll;
```

其中, `visible` 表示内容不会裁剪,但会呈现在元素之外; `hidden` 表示溢出的内容自动隐藏; `auto` 表示自适应,在需要时再产生滚动条; `scroll` 表示始终显示滚动条,溢出内容裁剪掉。

- `list-style`: 设置列表项的样式,常用它消除列表项前的小黑点,语法如下:

```
list-style:none;
```

- `border-collapse`: 设置表格显示方式,语法如下:

```
border-collapse:separate |collapse;
```

其中, `separate` 是默认方式,这两者的区别可以使用例 2-8 中的表格测试,对 `table` 标签设置 `border-collapse` 的效果如图 3-57 所示。

存款时间	存款利率
一年	1.75
二年	2.25
三年	2.75
五年	2.75
注: 数据来源于中国工商银行	
seperate	

存款时间	存款利率
一年	1.75
二年	2.25
三年	2.75
五年	2.75
注: 数据来源于中国工商银行	
collapse	

图 3-57 对 `table` 标签设置 `border-collapse` 的效果

- `-webkit-tap-highlight-color`: 用于设定元素在移动设备(如 Android、iOS)上被触发点击事件时,响应的背景框的颜色。一般设置为透明色,防止元素被单击时出现背景色。

```
-webkit-tap-highlight-color: transparent;
```

- `-webkit-user-select`: 也是 App 开发中常用的,用于设置是否允许用户选中文本,它的设置一般为:

```
-webkit-user-select: none|text;
```

其中,none 表示不允许选中文本,text 表示可以选择。

3.13 移动设备的适配

除了在开发中在 head 标签加入 `< meta name="viewport" .../>` 以外,HTML5 页面对于移动设备的适配还需要其他方法。

1. Media Queries

Media Queries 又叫媒体查询方法,它能在不同的条件下使用不同的样式,使页面在不同终端设备下达到不同的渲染效果。这里具体说明其使用方法,在内嵌样式或链接样式中使用它,语法形式如下:

```
@media 媒体类型 and (媒体特性){CSS 样式设置列表}
```

使用 Media Queries 必须要使用“@media”开头。媒体类型一般都使用 screen,表示用于电脑屏幕、平板、智能手机。媒体特性的书写方式和样式的书写方式非常相似,主要分为两个部分,第一个部分指的是媒体特性,第二部分为媒体特性所指定的值。但与 CSS 属性不同的是,媒体特性是通过 min/max 来表示大于等于或小于作为逻辑判断,而不是使用小于号(<)和大于号(>)来判断。常用的媒体特性见表 3-11。

表 3-11 常用的媒体特性

媒体特性	说 明
aspect-ratio	设备中的页面可见区域宽度与高度的比率
device-aspect-ratio	设备的屏幕可见宽度与高度的比率
device-height	设备的屏幕可见高度
device-width	设备的屏幕可见宽度
max-device-aspect-ratio min-device-aspect-ratio	输出设备屏幕可见宽度与高度的最大(小)比率
max-device-height min-device-height	输出设备的屏幕可见的最大(小)高度
max-device-width min-device-width	输出设备的屏幕最大(小)可见宽度
max-resolution max-resolution	设备的最大(小)分辨率

多个媒体特性使用关键字 and 连接,例如下面是目前流行的 iphone6 和 iphone6+ 的 Media Querires 写法:

```
@media only screen and (min-device-width: 375px) and (max-device-width: 667px) and (orientation: portrait) {
```

```

//iPhone 6 竖屏
}

@media only screen and (min-device-width: 375px) and (max-device-width: 667px) and
(orientation: landscape) {
    //iPhone 6 横屏
}

@media only screen and (min-device-width: 414px) and
(max-device-width: 736px) and (orientation: portrait) {
    //iPhone 6+ 竖屏
}

@media only screen and (min-device-width: 414px) and
(max-device-width: 736px) and (orientation: landscape) {
    //iPhone 6+ 横屏
}

@media only screen and (max-device-width: 640px), only screen and (max-device-width:
667px),
only screen and (max-width: 480px){
    //iPhone 6 and iPhone 6+ 横、竖屏
}

```

only 用来指定某种特定的媒体类型,可以用来排除不支持媒体查询的浏览器。其实,only 很多时候是用来对那些不支持 Media Query 的设备隐藏样式表的。

2. 百分比布局

HTML 中布局元素的宽度不再写成绝对值,而是以容器的宽度作为标准,换算成所占百分比,这样无论屏幕宽度大小如何变换,布局结构不会发生变化,如图 3-58 所示的布局中,页面内容宽度为 1000px,各部分宽度(包括外边距)按照百分比计算方式。

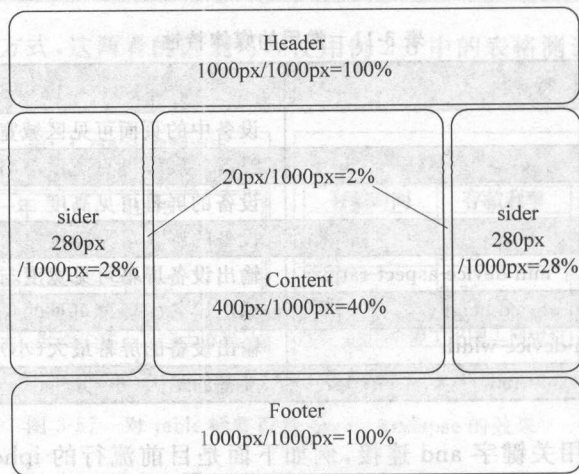


图 3-58 百分比布局

但这种方式对于高度的设置必须要求父容器的高度有明确的值,所以必须为 body 设置高度 height:100%。

3. rem 布局

所谓 rem 布局,是指以 html 标签元素定义 font-size 为基础,例如:

```
html{font-size:10px;}
```

那么将有 $1\text{rem}=10\text{px}$, $1.1\text{rem}=11\text{px}$, 所有的大小、边距等都以 rem 作为单位, CSS 设置时需要换算成 rem 设置, 例如:

```
#div1{margin-top:0.5rem} //上外边距为 5px, 使用 0.5rem
```

这样结合 Media Queries 或 JavaScript 技术, 将 html 中的 font-size 根据不同的移动设备进行动态设置, 就可以达到适配的要求。



HBuilder 中已内置了 px 转 rem 自动提示的功能, 需要自行配置, 配置方法请自行参考 <http://ask.dcloud.net.cn/article/982>。

3.14 实例

3.14.1 注册表单样式美化

【例 3-8】注册表单样式美化。

例 2-9 完成了一个简单的注册页面, 但是界面看起来差强人意, 在这里, 我们利用学过的 CSS 属性, 为界面的表单进行了样式美化。

由于 CSS 代码较多, 就不在此处赘述了, 请参考本书的配套源代码, 结合 Chrome 的“开发者工具”学习, 在这个基础上, 可以设计出更漂亮的注册界面。本例的运行效果如图 3-59 所示。

欢迎注册我们的网站!

用户名: 请输入用户名

密码: 请输入密码

确认密码: 请确认密码

区域: 四川省

性别: ☒ 男 ☐ 女

年龄: 20

生日: 年 / 月 / 日

手机号:

头像: 未选择任何文件

主页:

Email:

喜欢的颜色:

☒ 同意服务条款

图 3-59 注册界面美化

3.14.2 旅游 App 页面

【例 3-9】 旅游 App 页面实现。由于 CSS 和 HTML 代码都较多,就不在此处赘述了,请参考本书的配套源代码,结合 Chrome 的“开发者工具”学习,浏览时请切换到移动设备模式。

页面采用 rem 布局实现的适配,html 的 font-size 属性的设置借助了 Media Queries,页面在 iphone5、iphone6、iphone6+、Galaxy S5 作了适配(其他设备的适配请读者自行练习),浏览的效果如图 3-60 所示。界面中元素的 CSS 属性单位都是使用 rem,没有使用 px。



图 3-60 旅游 App 页面

小结

本章主要学习了 CSS 样式设计页面中的使用,讲解了 CSS 样式规则、颜色和单位,如何在页面中应用 CSS,以及 CSS 的各种选择器、CSS 的各种属性的使用,包括尺寸、文本、背景、边框、动画。对于 CSS 中较难的知识,包括 CSS 层叠性和优先级、盒子模型、浮动与定位,移动设备的适配都作了详细讲解。限于篇幅,本章无法对所有的 CSS 属性都作具体的讲解,请自行参考其他书籍。

习题

一、选择题

- 下面()项是 CSS 的正确语法。
 - body:color = black
 - {body;color:black;}
 - body{color:red;}
 - body{color = black;}
- 以下用于给页面所有 h1 标签添加背景色的是()。
 - h1{background-color:"red";}
 - #h1{background-color:"red";}
 - h1 all{background-color:"red";}
 - .h1{background-color:"red";}
- 下面()CSS 属性设置可以设置 HTML 元素的内填充左、上、右、下分别是 10px、20px、30px、40px。
 - padding:10px 20px 30px 40px
 - padding:20px 30px 40px 10px
 - padding:30px 40px 10px 20px
 - padding:40px 10px 20px 30px
- 页面上的 div 标签设计和为它设计的样式如下,这个 div 的背景色是()。

```

<style>
  div{
    width: 100px;
    height: 100px;
    background-color: yellow;
  }
  #div1{
    background-color: blue!important;
  }
  .fordiv{
    background-color: green;
  }
</style>

<div style="background-color: red;" id="div1" class="fordiv"></div>

```

- blue
 - yellow
 - green
 - red
- 下列()样式设置后,行内元素可以定义宽度和高度。
 - display:inline;
 - display:none;
 - display:block;
 - display:inherit;
 - 要实现一张扑克牌在桌面上的翻牌动作,可以使用 CSS 变换中的()方法。
 - rotate()
 - rotateX()
 - rotateY()
 - skew()
 - a:hover 表示超链接在鼠标()的状态。
 - 按下去
 - 经过
 - 悬停
 - 访问后
 - 在 Webkit 核心的浏览器中,设置 transform 属性需要添加私有前缀()。
 - ms-
 - o-
 - webkit-
 - moz-

二、判断题

1. 相对定位中,HTML 标签元素原来所占有的空间会被保留。()
2. 内填充和外边距的值不能为负值。()
3. div 是行内元素,span 是块级元素。()
4. z-index 可用于设置 HTML 元素的重叠定位顺序,值越大的越在上面。()
5. 过渡属性 transition 不能设置动画逆向。()
6. 父级元素的 position 属性可以被子元素继承。()

三、填空题

1. 在页面中使用 CSS 有三种方式,分别是_____、_____、_____。
2. CSS 的 id 选择器要在定义的前面使用符号 _____, class 选择器使用符号 _____。
3. HTML 标签元素的绝对定位是相对于_____实现定位的。
4. 向左浮动可以使用 CSS 属性 _____,元素两边都不允许有浮动,应该书写 CSS 属性为_____。
5. 为 HTML 标签元素设置边框圆角需要使用 CSS 属性_____。
6. 使用网络字体应该使用 CSS 规则_____。

四、简答题

1. CSS 的层叠性是如何体现的? 什么是优先级?
2. 简述 CSS Sprite 的原理及使用技巧。

五、编程题

完成一个纯 CSS 实现的下拉菜单效果,如图 3-61 所示。



图 3-61 CSS 下拉菜单效果

第 4 章

CHAPTER 4

JavaScript 编程基础

学习目标

- 掌握 JavaScript 在页面中的使用。
- 掌握 JavaScript 的基础语法和调试技巧。
- 掌握函数的使用。
- 掌握 JavaScript 的各种内置对象以及实现自定义类和对象。
- 掌握 JSON 数据格式以及序列化和反序列化。

HTML5 App 的开发都是基于 JavaScript 的,因此掌握 JavaScript 语言至关重要。本章主要讲解 JavaScript 在 App 开发中需要掌握的一些编程基础,在学习过程中,可与其他编程语言作对比学习,以便更快掌握这门语言。

4.1 JavaScript 介绍

JavaScript 诞生于 1995 年,当时走在技术革新最前沿的 Netscape(网景)公司,决定着手开发一种客户端语言,用来处理 Web 页面的验证,所以 Netscape 与 Sun 公司成立了一个开发联盟,联合开发出了 LiveScript,为了搭上当时媒体热炒 Java 的顺风车,就把 LiveScript 更名为 JavaScript。所以,从本质上来说,JavaScript 和 Java 没什么关系。

JavaScript 1.0 获得了巨大的成功,Netscape 随后在 Netscape Navigator 3(网景浏览器)中发布了 JavaScript 1.1。之后,作为竞争对手的微软在其 IE3 中加入了名为 JScript(名称不同是为了避免侵权)的 JavaScript 实现。这意味着此时市面上有 3 个不同的 JavaScript 版本,IE 的 JScript、网景的 JavaScript 和 ScriptEase 中的 CEnvi。随着版本不同所暴露的问题日益加剧,JavaScript 的规范化最终被提上日程。

1997 年,以 JavaScript 1.1 为蓝本的建议被提交给了欧洲计算机制造商协会(ECMA, European Computer Manufactures Association),该协会指定 39 号技术委员会负责将其进行标准化,TC39 由各大公司以及其他关注脚本语言发展的公司的程序员组成,经过数月的努力,完成了 ECMA-262——定义了一种名为 ECMAScript 的新脚本语言的标准。第二年,ISO/IEC(国标标准化组织和国际电工委员会)也采用了 ECMAScript 作为标准(即 ISO/IEC-16262)。ECMAScript 定义了 JavaScript 的标准,并不与任何浏览器相绑定,主要描述语法、类型、语句、关键字、保留字、运算符、对象等,在 HTML5 App 中主要应用的就是

ECMAScript 的语法。

JavaScript 具有以下特点：

- (1) 脚本语言,采用小程序段的方式进行编程。它的基本结构和 Java、C# 类似,但它是解释性语言,不需要先编译,而是由浏览器内核负责解释执行。
- (2) 跨平台性,只依赖于浏览器内核,与操作系统无关。
- (3) 它是基于对象(Object Based)和事件驱动(Event Driver)的编程语言,本身提供了很丰富的内部对象。

4.2 使用 JavaScript

目前网页上很多特效都是基于 JavaScript 实现的。在移动互联网时代,JavaScript 也是大放异彩,所有的 HTML5 应用都是基于 JavaScript 开发的,别外 JavaScript 语言也可以用于 Unity3D 游戏开发。

本章的 JavaScript 语言内容主要针对在 HTML5 App 中开发时所要用到的一些基础知识进行讲解,不过多涉及在浏览器中需要用到的知识。

4.2.1 在页面中插入代码

在 HTML 文件中使用 JavaScript 脚本时,JavaScript 代码需要出现在<script>标签和</script>标签之间。<script>标签的作用是通知浏览器内核,该位置是 JavaScript 代码,请负责解释执行,如果忘了<script>标签,所有的代码会作为文本输出到页面上。旧的网页标准要求标签中加入 type="text/javascript" 属性,在 HTML5 规范中,这个属性已成为默认值,没必要再添加。由于 JavaScript 语言是解释型,理论上它放置在 HTML 页面中的任一位置都可以运行。

【例 4-1】 在 HTML 文件中使用 JavaScript 的实例,代码如下:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="initial-scale=1.0,
    maximum-scale=1.0, user-scalable=no" />
  <title>使用 JavaScript</title>
</head>
<body>
  <script>
    alert("hello");
  </script>
</body>
```

新建移动 App,在页面中输入代码后,启动 Chrome 运行,结果界面如图 4-1 所示,这就是常见的网页对话框效果。可以尝试将<script>代码复制到页面的任一位置,你会发现程序都能运行。但是如果不小心出现了语法错误,例如将 alert 不小心输成 aler,会发现 HBuilder IDE 以及浏览器都不会有错误提示,这是 JavaScript 这种解释性语言的特性。当

然，IDE 的智能提示在很大程度上都会避免类似的错误。

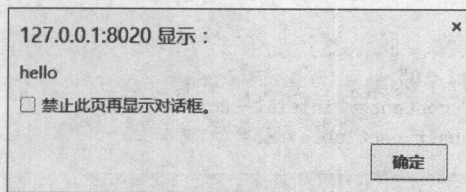


图 4-1 对话框效果



`<script>` 标签应尽量放置在 `body` 标签底部，`</body>` 结束标签之前。

4.2.2 使用 js 文件

另外一种使用 JavaScript 程序的方法是把 JavaScript 代码保存中一个 .js 文件中，然后在 HTML 文件中引用该 .js 文件，同样也是使用 `<script>` 标签，方法如下：

```
<script src = ".js 文件相对目录"></script>
```



在 HBuilder 中，在 `script` 标签输入 `src` 属性后，HBuilder 会智能提示 .js 的路径，只需选中代码便能自动完成，使用非常方便。

【例 4-2】 在 HTML 文件中使用 .js 文件的实例。

选中 App 项目的 js 目录，单击鼠标右键，打开“新建”菜单，选择“JavaScript 文件”命令，弹出对话框如图 4-2 所示，其中，在“选择模板”选项中选择“空白文件”，输入文件名 `test.js`，单击“完成”按钮。

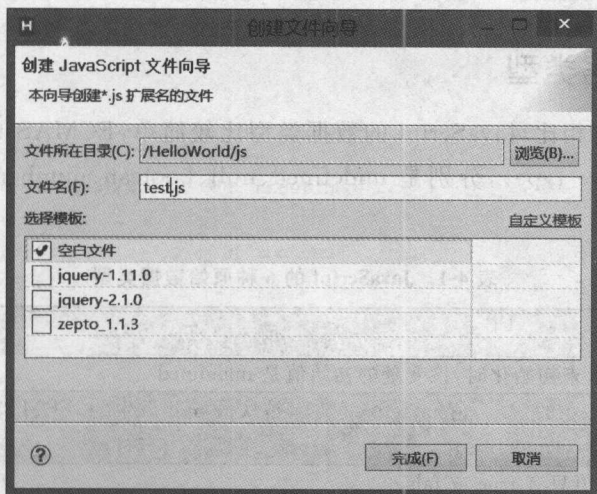


图 4-2 “创建文件向导”对话框

在 `test.js` 文件中，输入例 4-1 中的代码：`alert(“hello”)`，再修改页面如下，运行后的效果和例 4-1 是一致的。

```
<!DOCTYPE html>
<html>
<head>
  <meta charset = "utf - 8">
  <meta name = "viewport" content = "initial - scale = 1.0,
    maximum - scale = 1.0, user - scalable = no" />
  <title></title>
</head>
<body>
  <script src = "js/test.js"></script>
</body>
</html>
```



在 .js 文件中是不需要使用< script>标签的；< script>标签在用于引入 .js 文件后，其内部不得再用于嵌入 JavaScript 代码。

下面的这种代码是错误的，运行后，会发现 alert(“world”)压根不执行。

```
<script src = "js/test.js">
  alert("world");
</script>
```

4.3 JavaScript 的基础语法

本节主要介绍 JavaScript 的基础语法，包括数据类型、变量定义、数据类型的转换、常用语句等。

4.3.1 数据类型

与其他编程语言相比,JavaScript 的数据类型比较简单,ECMAScript 中定义了 6 种原始数据类型(primitive type),分别是 undefined、null、boolean、number、string 和 object,如表 4-1 所示。

表 4-1 JavaScript 的 6 种原始数据类型

数据类型	描 述
undefined	声明的变量未初始化时,该变量的初始值是 undefined
null	用于尚未存在的对象,值 undefined 实际是从值 null 派生的,ECMAScript 把它们定义为相等,即 null==undefined
boolean	布尔类型,值只有 true 或 false
number	数值类型,整数或浮点数
string	字符串类型,和其他语言不同,既可以使用单引号,也可以使用双引号
object	对象类型

4.3.2 变量定义

与其他编程语言不同的是,JavaScript 没有过多复杂的变量声明语句,所有的变量都只需要使用一个关键字 `var` 声明(C#在.NET 3.5 以后也借鉴了这个关键字),声明时可以赋值,也可以不声明而直接使用变量,这也是合法的,例如:

```
var x = 1;    //或修改成 x = 1;
alert(x);
```

JavaScript 变量定义必须遵守 2 条规则:

- 第一个字母必须是字母、下画线(_)或美元符号(\$);
- 其他字符可以是下画线、美元符号或任何字母或数字字符。



为了养成良好的编码习惯,在使用变量前必须声明,并且使用一定的编码规则,例如 Pascal 标记法等,变量名是区分大小写的;一个 `var` 只定义一个变量;每行只放一条 JavaScript 语句。

JavaScript 的变量还有一点与其他编程方式不同,它的变量是弱类型的,这意味着它可以随时根据所存储的值,自动修改变量的数据类型。

【例 4-3】 JavaScript 变量自动切换数据类型示例,使用了 `typeof` 运算符,代码如下:

```
<!DOCTYPE html >
<html >
<head>
  <meta charset = "utf - 8">
  <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
  <title>JavaScript 变量自动切换数据类型</title>
</head>
<body>
  <script>
    var x;
    alert(typeof(x)); //输出"undefined"
    x = true;
    alert(typeof(x)); //输出"boolean"
    x = "hello";
    x = 'hello';
    alert(typeof(x)); //输出"string"
    x = 12;
    x = 12.5;
    alert(typeof(x)); //输出"number"
    x = null;
    alert(typeof(x)); //输出"object"
  </script>
</body>
</html>
```


typeof 是一元运算符,它返回的结果始终是一个字符串,对于不同的操作数,它返回不同的结果。表 4-2 列出了它的不同返回结果。

表 4-2 typeof 的不同返回结果

变量的值	typeof 的返回结果
数字类型	"number"
字符串类型	"string"
布尔类型	"boolean"
对象、数组、null	"object"
函数名字	"function"
未定义的变量或未赋值的变量	"undefined"

4.3.3 数据类型的转换

在编程过程中,经常涉及一些数据类型的转换,例如数字转字符串、字符串转数值等,JavaScript 提供了一套转换机制,包括使用强制类型转换、转换函数、隐式类型转换。表 4-3 总结了 JavaScript 的每一种转换,并且针对每一种特定类型的值,给出了所执行的转换结果。

表 4-3 JavaScript 类型转换表

值	字符串	数字	布尔值	对象
未定义的值	"undefined"	NaN	false	Error
null	"null"	0	false	Error
非空字符串	不变	字符串中的数字值或 NaN	true	String 对象
空字符串	不变	0	false	String 对象
0	"0"	不变	false	Number 对象
NaN	"NaN"	不变	false	Number 对象
无穷大	"Infinity"	不变	true	Number 对象
负无穷大	"-Infinity"	不变	true	Number 对象
其他数字	数字的字符串值	不变	true	Number 对象
true	"true"	1	不变	Boolean 对象
false	"false"	0	不变	Boolean 对象

对于表 4-3 中的几个关键词解释如下:

(1) NaN: JavaScript 的一个特殊值,用于表示变量或结果不是数字,在 JavaScript 中可以用 isNaN() 全局函数来判断一个值是否是 NaN 值,例如:

```
var Month = 30;
if(Month < 1 || Month > 12)
{
    Month = NaN;
}
alert(Month);    //输出"NaN"
```

```
var x = "hello";
alert(isNaN(x));    //输出"true"
```



JavaScript 的语法与 C、Java、C# 很类似，代码块要采用 { } 包含起来；另外结尾 “;” 可以省略，但一般公司的编码规范都强制要求必须使用 “;”。

(2) Infinity: JavaScript 中用于存放无穷大的数值，-Infinity 则存放负无穷大，超出 $1.7976931348623157E+10308$ 的数值即为 Infinity，小于 $-1.7976931348623157E+103088$ 的数值为无穷小，例如下面的代码：

```
var x = 1.7976931348623157E+10308;
alert(x);    //输出"Infinity"
var y = -1.7976931348623157E+10308;
alert(y);    //输出"- Infinity"
alert(3/0);  //输出"Infinity"
```

1. 强制类型转换

ECMAScript 中可用的有 3 种强制类型转换方法。

(1) Boolean(): 把给定的值转换成 Boolean 类型，例如：

```
alert(Boolean(3));    //输出 true;
alert(Boolean(0));    //输出 false;
alert(Boolean(-1));   //输出 true;
alert(Boolean("true")); //输出 true;
alert(Boolean("false")); //输出 true;
alert(Boolean(""));    //输出 false;
alert(Boolean(null));  //输出 false
```

(2) Number(): 把给定的值转换成数字(可以是整数或浮点数)，例如：

```
alert(Number(false)); //输出 0
alert(Number(true));  //输出 1
alert(Number(undefined)); //输出 NaN
alert(Number(null));  //输出 0
alert(Number("5.5")); //输出 5.5
alert(Number("56"));  //输出 56
alert(Number("5.6.7")); //输出 NaN
```

(3) String(): 把给定的值转换成字符串，可把任何值转换成字符串，与调用 toString() 方法的唯一不同之处在于，对 null 或 undefined 值强制类型转换可以生成字符串而不引发错误，例如：

```
alert(String(null)); //输出"null"
var x;
```



```

alert(String(x));           //输出"undefined"
alert(String(true));        //输出"true"
alert(String(false));       //输出"false"
alert(String(Infinity));    //输出"Infinity"

```

2. 转换函数

(1) toString(): 把给定的值转换成字符串,还可以得到数字相应的进制转换数据,例如:

```

var x = true;
var y = x.toString();
alert(typeof(y));           //输出 string
alert(x.toString());         //会引发错误
alert(undefined.toString()); //会引发错误
var z = 8;
alert(z.toString("2"));      //输出 8 的二进制数据字符串"1000"

```

(2) parseInt(): 把给定的值转换成整数。

在判断字符串是否是数字值之前,都会仔细分析该字符串:首先查看位置0处的字符,判断它是否是有效数字,如果不是,方法将返回 NaN,不再继续执行其他操作。如果该字符是有效数字,该方法将查看下一位置的字符,进行同样的测试。这一过程将持续到发现非有效数字的字符为止,此时 parseInt() 将把该字符之前的字符串转换成数字,例如:

```

var x = "123";
alert(parseInt(x));          //输出 123
var y = "454px";
alert(parseInt(y));          //输出 454
var z = "22.5";
alert(parseInt(z));          //输出 22
var h = "blue";
alert(parseInt(h));          //输出 NaN

```

parseInt 默认实现是十进制,它还可以进行其他进制的转换,例如:

```

var x = "AF";
alert(parseInt(x,16));       //输出 175
var y = "10";
alert(parseInt(y,2));        //输出 2
alert(parseInt(y,8));        //输出 8
alert(parseInt(y,16));       //输出 16

```

(3) parseFloat(): 把给定的值转换成浮点数。

与 parseInt() 方法的处理方式相似,也是从位置0开始查看每个字符,直到找到第一个非有效的字符为止,然后把该字符之前的字符串转换成数字。不过,不同的是,对于第一个

出现的小数点是有效字符。如果有两个小数点,第二个小数点将被看作是无效的,parseFloat()方法会把这个小数点之前的字符串转换成数字。另一不同之处在于,parseFloat()也没有进制转换模式,字符串必须以十进制形式表示浮点数,而不能用八进制形式或十六进制形式。

```
alert(parseFloat("1234blue"));    //输出 1234.0
alert(parseFloat("0xA"));         //输出 0
alert(parseFloat("22.5"));        //输出 22.5
alert(parseFloat("22.34.5"));     //输出 22.34
alert(parseFloat("0908"));        //输出 908
alert(parseFloat("blue"));        //输出 NaN
```



HBuilder 在代码的智能提示方面的表现非常棒,而且还会贴心提示该语法在浏览器中的支持情况(包括移动端 Android 和 iOS),并提供了使用示例,如图 4-3 所示。



图 4-3 HBuilder 贴心的智能提示

3. 隐式类型转换

对于隐式类型转换的良好掌握,在实际应用中能够简化很多操作,例如:

```
var x = 8;
alert(x + "hello");
//输出 8hello, 将数字先转换为字符串, 然后进行字符串连接
var y = true;
alert(x + y);    //输出 9, 先将布尔值转成数字, 然后进行相加
var z = "6";
alert(z - y);    //输出 5, 字符串和布尔值都转成数字相减
```

4.3.4 代码注释

代码注释是程序代码中不执行的文本字符串,主要用于代码说明,或暂时禁用。使用注

释,可使程序代码更易于理解和维护。注释通常用于说明代码的功能,描述复杂计算或解释思路,记录程序名、作者名、主要代码更改的日期等。JavaScript 支持 2 种注释字符。

1. 单行注释//

//是单行注释符号,这种注释可与代码同一行,也可以另起一行。从//开始到行尾均表示注释。在 HBuilder 中可以使用鼠标选中要注释的代码行,按 Ctrl+/快捷键实现代码单行注释,再按一次,可以去除注释。前面的代码示例中已经多次使用了这种注释方式。

2. 多行注释/* ... */

/* ... */是多行注释符号,...表示要注释的内容。这种注释同样也可与代码一行,也可以另起一行。对于多行注释,必须使用开始注释符(/*)开始注释,使用结束注释符(*/)结束注释。注释行上不应出现其他注释字符。在 HBuilder 中可以使用鼠标选中要注释的代码行,按 Ctrl+Shift+/快捷键实现代码多行注释,再按一次,可以去除注释。下面是个多行注释的示例,一般各公司都有自己的编码规范,对代码的注释也会作详细的规定。

```
/* Description:下面的这些代码会输出一个标题和一个段落
 * Author:黄波
 * Date:2016-08-25
 */
var x = 8;
...
```

4.3.5 运算符

运算符可以指定变量和值的运算操作,是构成表达式的重要元素。JavaScript 支持一元运算符、算术运算符、位运算符、赋值运算符、关系运算符、逻辑运算符、条件运算符等基本运算符。本节分别对这些运算符作简单介绍。

1. 一元运算符

- delete: 删除自定义的对象属性及方法的引用,例如:

```
var o = new Object();
o.name = "huangbo";
alert(o.name);//输出"huangbo"
delete o.name;
alert(o.name);//输出 undefined
```

- void: 对任何值都返回 undefined,通常用于避免输出不应该输出的值,例如在页面设计中,经常需要一个空链接时,就可以利用 void,例如:

```
<a href = "javascript:void(0);">hello</a>
```

- ++: 增量运算符,分前增量和后增量(以放在操作变量前或后来划分),前者表示在使用变量时,变量先加 1,后者表示变量使用完后,变量再加 1,例如:

```
var x = 5;
alert(x++); // 输出 5, x 变为 6
alert(++x); // 输出 7
```

- **--**: 减量运算符, 分前减量和后减量(以放在操作变量前或后来划分), 前者表示在使用变量时, 变量先减 1, 后者表示变量使用完后, 变量再减 1, 例如:

```
var x = 5;
alert(x--); // 输出 5, x 变为 4
alert(--x); // 输出 5
```

- **+**: 一元加法, 可以理解为正号, 对数字无任何影响, 但对字符串使用会自动转成数字, 例如:

```
var x = 5;
alert(+x);           // 输出 5
var y = "5";
alert(typeof(+y));    // 输出 "number"
```

- **-**: 可以理解为负号, 对数字是取负, 但对字符串使用会自动转成数字再求负。

2. 算术运算符

算术运算符主要用于实现数学计算, 包括加(+)、减(-)、乘(*)、除(/)和求余(%), 这里面比较特殊的是除法和求余: 当数值除以 0 时, JavaScript 和其他语言不一样, 不会报错或抛出异常, 相反地, 会输出 Infinity; 求余表示左操作数除以右操作数后所得的余数, 例如:

```
alert(3/0);           // 输出 "Infinity"
alert(10%6);           // 输出 4
alert(6%9);           // 输出 6
```

3. 位运算符

ECMAScript 整数有两种类型, 即有符号整数(允许用正数和负数)和无符号整数(只允许用正数)。在 ECMAScript 中, 所有整数字面量默认都是有符号整数, 这意味着有符号整数使用第 31 位表示整数的数值, 用第 32 位表示整数的符号, 0 表示正数, 1 表示负数。数值范围为 -2^{31} ~ $2^{31}-1$ 。可以以两种不同的方式存储二进制形式的有符号整数, 一种用于存储正数, 另一种用于存储负数。正数是以二进制形式存储的, 前 31 位中的每一位都表示 2 的幂, 从第 1 位(位 0)开始, 表示 2^0 , 第 2 位(位 1)表示 2^1 。没用到的位用 0 填充, 即忽略不计。图 4-4 展示的是数字 18 的表示法。

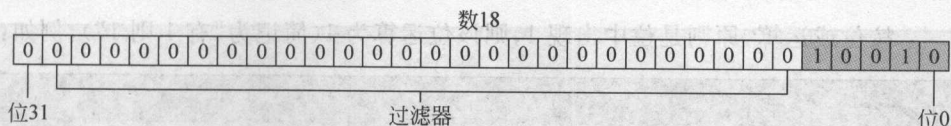


图 4-4 数字 18 的表示法

数字 18 的二进制只用了前 5 位,它们是这个数字的有效位。把数字转换成二进制字符串,就能看到有效位:

```
var x = 18;
alert(x.toString("2")); //输出 10010
```

负数也存储为二进制位,不过采用的是二进制补码,一个数字的补码计算可分为三步,还是以 -18 为例:

- ① 确定该数字的非负版本的二进制表示;

```
0000 0000 0000 0000 0000 0000 0001 0010
```

- ② 求得二进制反码,即要把 0 替换为 1,把 1 替换为 0;

```
1111 1111 1111 1111 1111 1111 1110 1101
```

- ③ 在进制反码上加 1。

```
1111 1111 1111 1111 1111 1111 1110 1101
                                     1
-----
1111 1111 1111 1111 1111 1111 1110 1110
```

因此, -18 的二进制表示为 1111 1111 1111 1111 1111 1111 1110 1110,在处理有符号的整数时,ECMAScript 规定不能访问第 31 位,把 -18 转换成二进制字符串后,并不是以二进制补码形式,而是用数字的绝对值的标准二进制代码前加负号的形式,例如:

```
var x = -18;
alert(x.toString("2")); //输出 -10010
```

- ~: 按位非运算,实质是对数字求负,再减 1,例如:

```
var x = 25;
alert(~x); //输出 -26
```

- &: 按位与运算,原则是位中出现 0,则该位运算为 0(简记为“有 0 则 0”),例如:

```
var x = 25&3;
alert(x); //输出 1
```

- |: 按位或运算,原则是位中出现 1,则该位运算为 1(简记为“有 1 则 1”),例如:

```
var x = 25|3;
alert(x); //输出 27
```


4. 赋值运算符

简单的赋值运算符由等号(=)实现,只是把等号右边的值赋予等号左边的变量。除此之外,每种主要的算术运算符以及其他几个运算符都可以与=组成复合赋值运算符:

- *=: 乘法/赋值;

```
var x = 10;
x * = 5;      //等效于 x = x * 5
alert(x);     //输出 15
```

- /=: 除法/赋值, $x /= 5$ 表示 $x = x / 5$;
- %=: 求余/赋值, $x \% = 5$ 表示 $x = x \% 5$;
- +=: 加法/赋值, $x += 5$ 表示 $x = x + 5$;
- -=: 减法/赋值, $x -= 5$ 表示 $x = x - 5$;
- <<=: 左移/赋值, $x <= 5$ 表示 $x = x < 5$;
- >>=: 有符号右移/赋值, $x >> = 5$ 表示 $x = x >> 5$;
- >>>=: 无符号右移/赋值, $x >>> = 5$ 表示 $x = x >>> 5$ 。

5. 关系运算符

关系运算符是对两个变量或数值进行比较,返回一个布尔值。JavaScript 的关系运算符有以下几种。

- ==: 等于运算符,和其他编程语言不同,为确定两个变量是否相等时,两个变量都会进行类型转换,例如:

```
alert(null == undefined);    //输出 true
alert("NaN" == NaN);        //输出 false
alert(false == 0);           //输出 true
alert(true == 1);            //输出 true
alert(true == 2);            //输出 false
alert("5" == 5);             //输出 true
```

- ===: 恒等运算符,和==不同的是,除了比较数值的相等,还要比较数据类型,例如:

```
var x = 6;
var y = "6";
alert(x == y);               //输出 true
alert(x === y);              //输出 false
```

- !=: 不等运算符;
- !==: 不恒等运算符,两个运算数在未进行类型转换之前是否不相等,例如:

```
var x = "5";
var y = 5;
```



```

alert(x!= y);    //输出 false
alert(x!== y);   //输出 true

```

- >: 大于运算符;
- <: 小于运算符;
- >=: 大于等于运算符;
- <=: 小于等于运算符。

6. 逻辑运算符

- &&: 逻辑与运算符,只要其中一个运算数为 false,则运算结果就为 false,参与逻辑运算的两个数可以是任何类型,不只是 Boolean 值,如果某个数不是 Boolean 值,该运算不一定返回 Boolean 值,例如:

```

var x = true;
var y = [1,2,3];
alert(x&& y);           //x 是 Boolean,y 是对象,输出 y 的值
var z = new Object();
alert(y&& z);            //y,z 都是对象,输出 z 的值
alert(x&& null);         //有运算数为 null,输出 null
alert(z&& NaN);          //有运算数为 NaN,输出 NaN
alert(x&& undefined);    //有运算数为 undefined,输出 undefined

```

- ||: 逻辑或运算符,只要其中一个运算数为 true,则运算结果就为 true,与 && 类似,如果某个数不是 Boolean 值,该运算不一定返回 Boolean 值,例如:

```

var x = true;
var y = [1,2,3];
var z = new Object();
alert(y|| z);           //y,z 都是对象,输出第一个对象的值

```

- !: 逻辑非运算符,运算结果一定是 Boolean 值,运算数不一定是 Boolean 值,例如:

```

var x = true;
alert(!x);              //输出 false
var y = new Object();
alert(!y);              //输出 false
alert(!0);              //输出 true
alert(!1);              //0 以外的数字,输出 false
alert(!null);           //输出 true
alert(!NaN);            //输出 true
alert(!undefined);      //输出 true

```

7. 条件运算符

条件运算符是 JavaScript 中功能最多的运算符,它的语法如下:

```
variable = boolean_expression? true_value:false_value;
```

该表达式主要根据 `boolean_expression` 的计算结果有条件地为变量赋值,如果 `boolean_expression` 为 `true`,则 `true_value` 赋给变量,如果为 `false`,则把 `false_value` 赋给变量,例如:

```
var x = 8;
var y = x > 10 ? 1 : 3;
alert(y);    //输出 3
```

4.3.6 常用语句

1. 条件语句

1) if 语句

if 语句是编程语言中最常用的语句之一,它的语法结构是:

```
if(条件)
{
    :
}
```

其中的条件可以是任何表达式,计算的结果甚至不必是真正的 Boolean 值,JavaScript 会把它转换成 Boolean 值,这点和其他编程语言是完全不同的,在学习时要特别注意这点。例如,下面这两段代码执行效果是一样的,但如果 `x=0`,则不会提示:

```
var x = 8;          var x = 8;
if(x > 3)           if(x)
{                   {
    alert("x 大于 3");    alert("x 大于 3");
}
```

在 HTML5 实际开发过程中,if 语句还会经常用来检测对象是否具有某个属性或某个方法是否存在,例如:

```
if(window.addEventListener){...}
//检测 winodw 对象是否有 addEventListener 方法
```



即使 if 后面只有一条语句,也应该使用 `{}` 代码块,这是一种良好的习惯,有些公司的编码规范里特别注明了这条。

2) if...else 语句

if...else 语句的语法结构是:

```

if(条件1){
    ...}
else{
    :
}

```

它主要用来实现两个分支,表示满足条件执行一段代码,不满足就执行另一段,例如:

```

var time = 17;
if(time < 20)
{
    alert("Good Day");
}
else
{
    alert("Good Evening");
}

```

3) if...else if...else 语句

else if 语句是 else 语句和 if 语句的组合,用来实现多分支,可以包含多个 else if 语句,例如下面这个例子:

```

var x = 4;
if(x == 1) {
    alert("星期一");
} else if(x == 2) {
    alert("星期二");
} else if(x == 3) {
    alert("星期三");
} else if(x == 4) {
    alert("星期四");
} else if(x == 5) {
    alert("星期五");
} else if(x == 6) {
    alert("星期六");
} else if(x == 7) {
    alert("星期日");
}

```



在 HBuilder 中,选中代码后,单击鼠标右键,选择整理代码格式,可以对代码迅速进行排版。

4) switch 语句

当有很多分支时,尽量还是使用 switch,代码会更简捷,它的语法结构如下:

```

switch(表达式)

```



```
{
    case 值 1:代码段 1
        break;
    case 值 2: 代码段 2
        break;
    case 值 3: 代码段 3
        break;
    :
    default:代码段
}
```

以 3) 中的代码为例,使用 switch 后,代码变成:

```
var x = 9;
switch(x) {
    case 1:
        alert("星期一");
        break;
    case 2:
        alert("星期二");
        break;
    case 3:
        alert("星期三");
        break;
    case 4:
        alert("星期四");
        break;
    case 5:
        alert("星期五");
        break;
    case 6:
        alert("星期六");
        break;
    case 7:
        alert("星期日");
        break;
    default:
        alert("x 的值不对");
        break;
}
```

2. 循环语句

循环语句主要用于声明一组要反复执行的代码,直到满足了某些条件为止。JavaScript 提供了以下语句。

1) while 语句

while 是先测试循环,满足条件才开始执行,退出条件是在执行循环内部代码之前计算的,while 语句的语法结构如下:

```
while(条件){
    ...
}
```

【例 4-4】 使用 while 语句来计算计算 $1+2+3+4+5+6+7+8+9+10$ 的结果,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>使用 while 循环</title>
  </head>
  <body>
    <script>
      var i = 1;
      var sum = 0;
      while(i < 11) {
        sum += i;
        i++;
      }
      alert(sum);
    </script>
  </body>
</html>
```

2) do...while 语句

do...while 是后测试循环,退出条件是在执行过循环内部的代码之后计算的。这意味着在测试循环条件之前,至少会执行一次循环,do...while 的语法结构如下:

```
do
{
    :
}while(条件);
```

【例 4-5】 将例 4-4 改写成 do...while 语句,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>使用 do-while 循环</title>
```



```

</head>
<body>
  <script>
    var i = 1;
    var sum = 0;
    do
    {
      sum += i;
      i++;
    }while(i<11);
    alert(sum);
  </script>
</body>
</html>

```

3) for 语句

for 循环是循环语句中使用频率最高的一个,它的语法结构如下:

```

for(表达式 1;表达式 2;表达式 3)
{
  :
}

```

代码在开始循环时计算表达式 1 的值,通常对循环计数器变量进行初始化设置;每次循环开始之前,计算表达式 2 的值,如满足,则继续循环,否则退出循环。每次循环结束之后,对表达式 3 进行求值,通常是用来改变循环计数器变量的值,使表达式 2 条件不满足,从而退出循环。

【例 4-6】 将例 4-4 改写成 for 语句,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title></title>
  </head>
  <body>
    <script>
      var sum = 0;
      for(var i = 1; i < 11; i++) {
        sum += i;
      }
      alert(sum);
    </script>
  </body>
</html>

```


4) for...in 语句

这是 JavaScript 提供了一种特殊的循环语句,用来迭代对象的属性或数组的每个元素,for...in 循环中的循环计数器是字符串,而不是数字。它包含当前属性的名称或当前数组元素的索引,例如:

```
for(var prop in window)
{
    alert(prop);    //遍历 window 对象的所有属性和方法
}
```

5) continue 语句

用于循环中需要跳过某次循环,继续下一次循环的情况。

【例 4-7】 计算 $1+2+3+5+6+7+8+9+10$,代码如下:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8">
        <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
        <title>使用 continue 语句</title>
    </head>
    <body>
        <script>
            var i = 0;
            var sum = 0;
            while(i < 10) {
                i++;
                if(i == 4) {
                    continue;
                }
                sum += i;
            }
            alert(sum);
        </script>
    </body>
</html>
```

6) break 语句

break 语句和它的英文含义一样,是用来直接退出循环的,阻止再次反复执行任何代码。

【例 4-8】 使用 break 语句修改例 4-4 的代码,代码如下:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8">
```

```

    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title>使用 break 语句</title>
</head>
<body>
    <script>
        var i = 1;
        var sum = 0;
        while(true) {
            sum += i;
            i++;
            if(i > 10) {
                break;
            }
        }
        alert(sum);
    </script>
</body>
</html>

```

4.4 函数

函数由若干条语句组成,用于实现特定的功能,一旦定义了函数,就可以在程序中需要实现该功能的位置调用该函数,给程序的复用带来了很方便。

4.4.1 函数定义及调用

函数是由关键字 function、函数名加一组参数以及置于花括号中的需要执行的代码段声明。函数的基本语法如下:

```

function functionName(arg0,arg1,...,argN) {
    //代码段
}

```

【例 4-9】 函数定义调用及示例,代码如下:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
        <title>函数定义调用</title>
    </head>
    <body>

```



```

<script>
    function sayHi(name,message)
    {
        alert("hello " + name + ", " + message);
    }
    sayHi("huangbo","how are you?");
</script>
</body>
</html>

```

这段代码会弹出一个简单的对话框,从这个例子可以看出,函数的参数前面不需要加任何关键字(这和 C 语言中的函数以及 Java、C# 的方法是完全不一样的)。



与其他语言不同,JavaScript 不会验证传递给函数的值是否与参数个数相等,函数可以接受任意个数的参数值,而不会报任何错误。任何遗漏的参数都会以 undefined 传递给参数,多余的参数将自动忽略。

【例 4-10】 JavaScript 的函数参数个数具有可变性,代码如下:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8">
        <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
        <title>JavaScript 的函数参数个数可变性</title>
    </head>
    <body>
        <script>
            function sayHi(name, message) {
                alert("hello " + name + ", " + message);
            }
            sayHi("huangbo", "how are you?"); //参数刚好匹配
            sayHi("huangbo"); //少一个参数值
            sayHi(); //没传值
            sayHi("huangbo","how are you?",true); //多传了一个参数值
        </script>
    </body>
</html>

```

4.4.2 变量的作用域

在函数中也可以定义变量,在函数中定义的变量称为**局部变量**。局部变量只在定义它的函数内部有效,在函数之外,即使使用同名的变量,也会被看作另一变量。相应地,在函数之外定义的局部变量是全局变量,它在当前整个页面中都是有效的。如果局部变量和全局变量同名,则在定义局部变量的函数中,只有局部变量是有效的。

【例 4-11】 以下是局部变量和全局变量作用域的例子,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>局部变量和全局变量作用域</title>
  </head>
  <body>
    <script>
      var a = 6;
      function test() {
        var a = 8;
        alert(a); //输出局部变量 8
      }
      test();
      alert(a);    //输出局部变量 6
    </script>
  </body>
</html>
```

4.4.3 函数重载

JavaScript 中的函数不能重载,可以使用相同的函数名在同一个作用域中定义两个函数,而不会引发错误,但真正使用的是最后一个函数,例如:

```
function doAdd(x) {
  alert(x + 10);
}
function doAdd(x) {
  alert(x + 100);
}
doAdd(50);
```

在函数代码中,JavaScript 提供了一个特殊对象 arguments,开发者无需明确指出参数名,就能访问它们,可以使用 arguments[0]访问第一个参数的值(第 1 个参数使用序号 0,第 2 个使用序号 1,依此类推),还可使用 arguments.length 来检测传递给函数的参数个数,利用 arguments 对象,可以模拟出函数重载,例如:

【例 4-12】 arguments 对象模拟函数重载,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
```

```
<meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
<title>arguments 对象模拟函数重载</title>
</head>
<body>
  <script>
    function doAdd() {
      if(arguments.length == 1) {
        alert(arguments[0] + 10);
      } else if(arguments.length == 2) {
        alert(arguments[0] + arguments[1]);
      }
    }

    doAdd(10);      //输出 20
    doAdd(20, 30);  //输出 50
  </script>
</body>
</html>
```

4.4.4 函数的返回值

可以为函数指定一个返回值,返回值可以是任何数据类型,使用 return 语句可以返回函数值并退出函数,例如:

```
function sum(x,y){
  return a+b;
}
var z = sum(2,3);
alert(z);    //输出 5
```



与其他语言不同,JavaScript 不要求代码所有执行路径都必须有返回值。如果函数没有明确的返回值,或调用了没有值的 return 语句,那函数的返回值就是 undefined。

【例 4-13】 函数返回值示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title>函数返回值</title>
  </head>
  <body>
```

```

<script>
    function doAdd(a, b) {
        if(arguments.length < 2) {
            return;
        }
        if(a && b && typeof(a) == "number" && typeof(b) == "number")
        {
            return a + b;
        }
    }
    var x = doAdd();
    alert(x);    //输出"undefined"
    var y = doAdd(2, 3);
    alert(y);    //输出"5"
    var z = doAdd(2, "hello");
    alert(z);    //输出"undefined"
</script>
</body>
</html>

```

这段代码中,求 z 的值时,doAdd 函数中两个 if 语句条件都不符合,所以实际执行中并没有遇到 return 语句返回值,所以 doAdd 的最终返回值为 undefined。

4.4.5 匿名函数

顾名思义,匿名函数就是没有实际名字的函数,通常用于需要立即执行的代码段或用作对象的方法上,例如下面这段代码,定义完一个匿名函数后立刻执行它:

```

(function(a, b) {
    alert(a + b);
})(2, 3);

```

匿名函数的作用是:

- 划出一块私有作用域,避免数据污染;
- 执行完就销毁,避免内存被一直占用。

4.5 JavaScript 在 Chrome 中的调试

4.5.1 在控制台输出

熟练使用 console.log,可以在 JavaScript 调试中省去不少麻烦,这个命令不像 alert 语句,必须由用户手动选择,它主要用于向控制台输出信息,而不会对界面或用户的操作产生任何影响,只需要在产品正式发布时注释掉即可。在 HBuilder 开发 App,连接真机调试时,这条语句也会自动向 HBuilder 的控制台输出相应信息。

【例 4-14】 以例 4-4 为例说明 console.log 的使用,代码修改如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>console.log 的使用</title>
  </head>
  <body>
    <script>
      var i = 1;
      var sum = 0;
      while(i < 11) {
        sum += i;
        console.log("i = " + i + ",sum = " + sum); //输出到控制台
        i++;
      }
      alert(sum);
    </script>
  </body>
</html>
```

启动 Chrome 后,弹出的结果和例 4-4 相同,控制台信息在哪里可以查看呢? 在 Chrome 中,按 F12 键,会自动调出 Chrome 的“开发者工具”,在选项卡上切换到 Console,就可以看到在控制台中控制输出的信息,也可以看到这个输出的语句行号,这在一定程度上为调试程序带来方便,如图 4-7 所示。

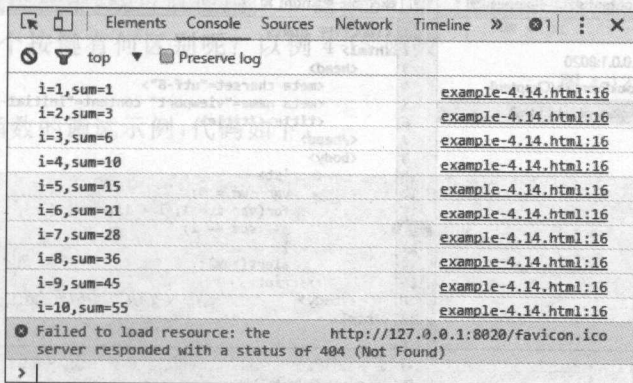



图 4-7 Chrome 控制台显示

 如果 JavaScript 在解释执行中发生错误,Chrome 控制台会以醒目的红色显示,控制台右上角也会以红色显示页面有几处错误。如例 4-14 所示,由于项目中没有自带 favicon.ico(浏览器收藏时显示的图标),控制台显示加载有问题(这个错误可忽略,因为只有 Web 项目一般才需要)。

如果把输出语句改成：

```
console.log("Hello");
```

Chrome 控制台会变成如图 4-8 所示的效果，这里 Hello 前面的 10 表示输出次数。

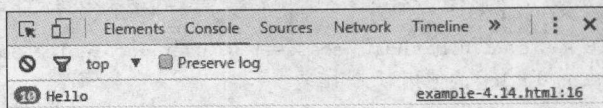


图 4-8 重复输出同一条语句

如果要清除控制台显示信息，可以在控制台中单击鼠标右键，选择 Clear console 选项。

4.5.2 断点调试

断点调试是指在 JavaScript 程序的某一行设置一个断点，调试时，程序运行到这一行就会自动停住，然后可以一步一步往下调试，调试过程中可以看程序是如何运行的，以及各个变量当前的值，如果出错，调试到出错的代码行即显示错误，程序自动停下。

1. 断点设置

现在以例 4-4 来说明在 Chrome 中如何实现调试：

- (1) 启动 Chrome，按 F12 键，打开“开发者工具”；
- (2) 单击选项卡上的“Sources”，打开 Sources 面板，展开路径，找到页面文件后单击；
- (3) 单击第 12 行左边的灰色区域，加上一个断点（再点一次会自动去除断点），如图 4-9 所示。

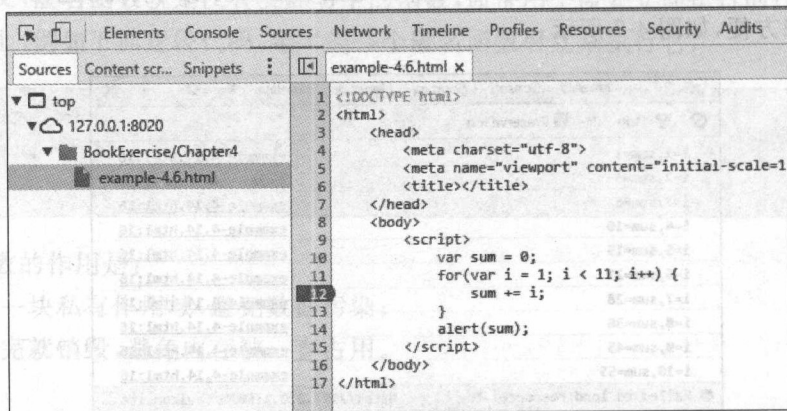


图 4-9 添加断点

(4) 按 F5 键刷新一次页面（因为在前面该页面已经加载完成），会发现程序自动停在断点这一行，这一行背景会自动变化，把鼠标悬停在 sum 变量上，会自动显示出 sum 变量当前的值，如图 4-10 所示。

(5) 按 F10 或 F11 键，控制程序一步步执行，随时观察 sum 变量的变化。



在 HBuilder IDE 中，选中代码左边有折叠符号的空白处，鼠标右键，

选择“显示行号”，可以迅速为代码添加行号显示。

2. 监视窗口的使用

在调试过程中，如果只能采用鼠标悬停对变量值进行观察会非常不方便，特别是遇到变量是个对象时，使用监视窗口会更方便，下面介绍它的使用。

(1) 代码停在断点处后，用鼠标选中要观察的变量 `sum`，单击鼠标右键，选择“Add to watch”命令，如图 4-11 所示。

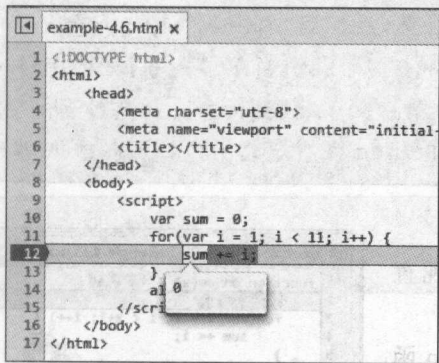


图 4-10 断点中断

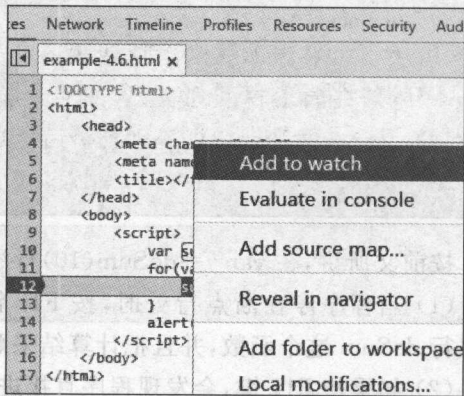


图 4-11 添加到监视窗口

(2) 对变量 `i` 作同样操作。

(3) 控制代码一次次执行，在工具的右侧监视窗口中，如图 4-12 所示，可以看到每次循环时 `i` 和 `sum` 的值的变化，在监视窗口中也可以看到断点所在行的位置。

3. 函数的调试

在前面的调试过程中，既可以使用 `F10` 键，也可以使用 `F11` 键，这两个按键有何区别呢？以例 4-15 函数的调试作为说明。

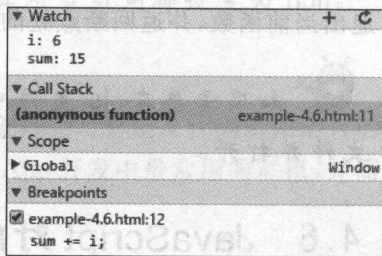


图 4-12 监视窗口

【例 4-15】 函数的调试示例，代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>函数的调试示例</title>
  </head>
  <body>
    <script src="../../js/example-4.15.js"></script>
    <script>
      var y = doSum(10);
      alert(y);
    </script>
  </body>
</html>
```



```

    </script>
  </body>
</html>

```

其中, example-4.15.js 的文件内容如下, 定义了一个函数 doSum:

```

function doSum(x) {
    var sum = 0;
    for(var i = 1; i < x+1; i++) {
        sum += i;
    }
    return sum;
}

```

按前文所讲, 在 `var y=doSum(10)` 行添加断点:

(1) 当程序停在断点行处时, 按 F10 键, 会发现直接执行 doSum 这个函数, 并且把计算结果赋给 y。

(2) 如果按 F11 键, 会发现程序直接进入 doSum 函数内部, 并将 10 赋值给参数 x, 如图 4-13 所示。

(3) 在调试函数的过程中, 可以随时按 Shift+F11 键, 退出当前函数, 并返回断点处的下一行语句。

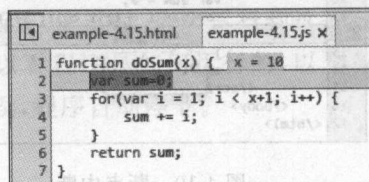


图 4-13 监视窗口



如果需要在 .js 文件中添加断点, 需要在 Sources 面板下找到相应的 .js 文件并打开。

4.6 JavaScript 对象基础

每个对象都由类定义, 类不仅要定义对象的属性和方法, 还要定义对象的内部工作(使用属性和方法发挥作用的代码)。编译器和解释程序都根据类的说明构建对象。程序使用类创建对象时, 生成的对象叫做类的实例。对类所生成对象的个数的唯一限制来自于运行代码机器的物理内存。每个实例的行为相同, 但实例处理一组独立的数据。由类创建对象实例的过程叫实例化(instantiation)。

JavaScript 并没有正式类, 没有像其他编程语言(如 C# 或 Java)中的关键字 class。

4.6.1 Object 对象

Object 对象, 是所有 JavaScript 对象的基类。它的声明如下, 下面这两句是等效的:

```

var myObject1 = new Object();
var myObject2 = {};

```

JavaScript 的对象与其他语言不同, 它的属性和方法可以动态附加到对象实例上, 属性


直接书写,方法使用函数 function 进行附加,例如:

```
var objStu = {};  
objStu.name = "huangbo";  
objStu.showStuInfo = function() {  
    alert(this.name);  
}  
objStu.showStuInfo();    //输出"huangbo"
```

JavaScript 中有无用存储单元收集程序,不必专门销毁对象来释放内存。当不再对对象进行引用时,称该对象被废除了。运行无用存储单元时,所有废除对象都会被销毁。例如,每当函数执行完,无用存储单元收集程序都会运行,释放所有的局部变量。

把对象的所有引用都设置为 null,可以强制性废除对象,例如:

```
var myObject = {};  
myObject = null;
```

 每用完一个对象,就将其废除来释放内存,这是好习惯。可以确保不再使用已经不能访问的对象,从而防止程序出错。但要注意,一个对象有两个或更多引用时,若要正确废除对象,必须将其所有引用都设置为 null。

4.6.2 内置对象

所有编程语言都具有内部(或内置的)对象。内部对象是编写自定义代码所用语言的基础,JavaScript 替代了丰富的内部对象。这里会介绍一些 App 开发中最常用的对象,并简要介绍它们的功能以及如何使用这些功能。

1. Math 对象

Math 对象主要用来处理一些常用的数学运算。Math 对象的常用方法如表 4-4 所示。

表 4-4 Math 对象的常用方法

方法	具体描述
abs	返回数值的绝对值
acos	返回数值的反余弦值
asin	返回数值的反正弦值
atan	返回数值的反正切值
atan2	返回由 X 轴到(y,x)点的角度(以弧度为单位)
ceil	返回大于等于其数字参数的最小整数
cos	返回数值的余弦值
exp	返回 e(自然对数的底)的幂
floor	返回大于等于其数字参数的最大整数
log	返回数字的自然对数
max	返回给出的两数值中的最大值
min	返回给出的两数值中的最小值

续表

方法	具体描述
pow	返回 x 的 y 次幂的值
random	返回介于 0~1 之间的伪随机数
round	把数值四舍五入为最接近的整数
sin	返回数字的正弦值
sqrt	返回数字的平方根
tan	返回数字的正切值

【例 4-16】 Math 对象的使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>Math 对象的使用</title>
  </head>
  <body>
    <script>
      console.log("求绝对值: " + Math.abs(-10)); //输出求绝对值: 10
      console.log("向上取整: " + Math.ceil(1.2)); //输出向上取整: 2
      console.log("四舍五入: " + Math.round(5.6)); //输出四舍五入: 6
      console.log("四舍五入: " + Math.round(5.4)); //输出四舍五入: 5
      console.log("取最大值: " + Math.max(1,8)); //输出取最大值: 8
      console.log("取最小值: " + Math.min(1,8)); //输出取最小值: 1
      console.log("生成随机数: " + Math.random()); //输出生成随机数: 0~1 的一个随机数
    </script>
  </body>
</html>
```

2. Date 对象

可以使用下面几种方法来创建 Date 对象,Date 对象的常用方法如表 4-5 所示。

```
var myDate = new Date();
var myDate2 = new Date("2016-09-01");
var myDate3 = new Date(2016,9,1);
```

表 4-5 Date 对象的常用方法

方 法	具体描述
getDate	从 Date 对象返回一个月中的某一天(1~31)
getDay	从 Date 对象返回一周中的某一天(0~6)
getMonth	从 Date 对象返回月份(0~11)
getFullYear	从 Date 对象以 4 位数字返回年份

续表

方 法	具 体 描 述
getHours	返回 Date 对象的小时(0~23)
getMinutes	返回 Date 对象的分钟(0~59)
getSeconds	返回 Date 对象的秒数(0~59)
getMilliseconds	返回 Date 对象的毫秒(0~999)
getTime	返回 1970 年 1 月 1 日至今的毫秒数
setDate	设置 Date 对象中月的某一天(1~31)
setMonth	设置 Date 对象中月份(0~11)
setFullYear	设置 Date 对象中的年份(四位数字)
setHours	设置 Date 对象中的小时(0~23)
setMinutes	设置 Date 对象中的分钟(0~59)
setSeconds	设置 Date 对象中的秒钟(0~59)
setMilliseconds	设置 Date 对象中的毫秒(0~999)
setTime	以毫秒设置 Date 对象
toString	把 Date 对象转换为字符串
toTimeString	把 Date 对象的时间部分转换为字符串
toDateString	把 Date 对象的日期部分转换为字符串

【例 4-17】 Date 对象的使用示例,代码如下:

```
<!DOCTYPE html >
<html >
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0,maximum - scale = 1.0, user - scalable = no" />
    <title> Date 对象的使用</title>
  </head>
  <body>
    <script>
      var myDate = new Date("2013 - 09 - 01 12:30:15");
      myDate.setFullYear(2016);
      myDate.setMonth(8);
      myDate.setDate(15);
      myDate.setHours(0);
      myDate.setMinutes(0);
      myDate.setSeconds(0);
      console.log(myDate.getFullYear() + "年" + (myDate.getMonth() + 1) + " 月"
        + myDate.getDate() + "日星期" + myDate.getDay()
        + " " + myDate.getHours() + "时" + myDate.getMinutes()
        + "分" + myDate.getSeconds() + "秒是中国的中秋佳节!");
      //输出"2016 年 9 月 15 日星期 4 0 时 0 分 0 秒是中国的中秋佳节!"
    </script>
  </body>
</html>
```

3. RegExp 对象

正则表达式是具有特殊语法的字符串,用来表示指定字符或字符串在另一个字符串中出现的情况。这些模式字符串,有的十分简单,有的十分复杂,它们可以实现很多功能,从删除字符串中的空格到验证 Email 格式的有效性等等。

JavaScript 对正则表达式的支持是通过 RegExp 类实现的,它是对字符串执行模式匹配的强大工具。正则表达式用法十分复杂,这里只简单介绍在 JavaScript 中如何使用 RegExp 来验证正则表达式。表 4-6 列出了一些常用的正则表达式字符串。

表 4-6 一些常用的正则表达式字符串

匹配方式	正则表达式字符串
中文	<code>[\u4e00-\u9fa5]</code>
Email	<code>\w+([-+. '\w+)*@ \w+([-+. '\w+)*\.\w+([-+. '\w+)*</code>
中国邮政编码	<code>[1-9]d{5}(?! d)</code>
Url	<code>http(s)?://([\w-]+ \.)+([\w-]+ /?%&=]*)?</code>
ip 地址	<code>^(25[0-5] 2[0-4][0-9] 1[0-9][0-9] [0-9]{1,2})\.(25[0-5] 2[0-4][0-9] 1[0-9][0-9] [0-9]{1,2}))\{3}\$</code>

RegExp 对象也可以直接定义,以字符“/”开始,接着是正则字符串,然后再以字符“/”结束。这两种方式是等价的,例如下面的代码定义了验证两位整数的正则,它们完全等价:

```
var reg = new RegExp("[1-2]\\d* $");
var reg1 = /[1-2]\d* $/;
```



如果使用字符串时,需要常规的字符转义规则,直接在字符前加“\”。

RegExp 提供了一个 test 方法,它会根据结果返回 true 或 false,代表是否通过验证,下面这个例子演示了如何使用 RegExp 来实现一些验证。

【例 4-18】 正则表达式验证的使用示例,一个是中文验证,另一个是 Email 验证,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>正则表达式验证的使用</title>
  </head>
  <body>
    <script>
      var reg1 = /[\u4e00-\u9fa5]/;
      alert(reg1.test("开发"));           //输出 true
      alert(reg1.test("HTML5 App"));      //输出 false
```

```
var reg2 = /\w+([-+.']\w+)*@\w+([-.\]\w+)*\.\w+([-.\]\w+)*;/;
var email1 = "abc@163.com";
var email2 = "abc@";
var email3 = "abc@163.";
alert(reg2.test(email1)); //输出 true
alert(reg2.test(email2)); //输出 false
alert(reg2.test(email3)); //输出 false
</script>
</body>
</html>
```

4. Array 对象

数组是内存中一段连续的存储空间,用于保存一组相同数据类型的数据。在JavaScript中定义数组可以用以下方法:

```
var myArray = new Array(); //定义个空数组
var myArray1 = new Array(3); //定义一个包含 3 个元素的数组
var myArray2 = new Array(1,2,3); //定义一个数组,并赋初值
```

在实际开发中,上面的代码一般都会使用方括号[]进行简化,例如:

```
var myArray = []; //定义个空数组
var myArray2 = [1,2,3]; //定义一个数组,并赋初值
```

数组的索引是从0开始的,可以通过索引访问数组元素,例如,可以通过 myArray[1]对数组的第2项来设置或读取值。

数组对象有一个属性 length,它代表数组的长度。和其他语言不同的是,JavaScript 数组动态数组的长度是可变的,例如下面代码,它的长度自动变为 26,中间的所有位置的数组元素的值自动全为 undefined,而对于数组的常用方法,则可以参看表 4-7:

```
var myColors = ["red", "green", "blue"];
alert(myColors.length); //输出长度 3
myColors[25] = "purple";
alert(myColors.length); //输出长度 26
alert(myColors[10]); //输出 "undefined"
```

表 4-7 数组对象的常用方法

方法	具体描述
concat	连接两个或更多的数组,并返回一个新的结果数组
join	把数组的所有元素连接成一个字符串,元素通过指定的分隔符进行分隔
push	入栈,向数组的末尾添加一个或更多元素
pop	出栈,删除并返回数组的最后一个元素

续表

方法	具体描述
reverse	颠倒数组中元素的顺序
shift	删除并返回数组的第一个元素
slice	返回子数组,分别指定开始和结束的索引
sort	对数组的元素进行排序
splice	从数组中删除指定个数的元素并返回新数组
unshift	向数组的开头添加一个或更多元素

【例 4-19】 Array 对象的使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale = 1.0,maximum-scale = 1.0, user-scalable = no" />
    <title>Array 对象的使用</title>
  </head>
  <body>
    <script>
      function shuffle() {
        var x = Math.random();
        var y = x > 0.5 ? -1 : 1;
        return y;
      }
      var aArray = [1, 2, 3];
      var bArray = [4, 5, 6];
      alert(aArray.concat(bArray));    //输出"1,2,3,4,5,6"
      alert(aArray);                  //不变,输出"1,2,3"
      alert(aArray.join("-"));        //输出"1-2-3";
      aArray.push(4);
      aArray.push(5, 6, 7);
      alert(aArray);                  //输出"1,2,3,4,5,6,7"
      alert(aArray.pop());            //输出"7"
      alert(aArray);                  //输出"1,2,3,4,5,6"
      alert(aArray.reverse());        //输出"6,5,4,3,2,1"
      alert(aArray.shift());          //输出"6"
      alert(aArray);                  //输出"5,4,3,2,1";
      alert(aArray.slice(1, 3));      //输出"4,3",不包括索引 3
      alert(aArray);                  //aArray 没变,输出"5,4,3,2,1";
      aArray.unshift(7, 6, 9, 8);
      alert(aArray);                  //输出"7,6,9,8,5,4,3,2,1"
      alert(aArray.sort(shuffle));    //将 aArray 进行随机打乱
      alert(aArray.sort());
      //从小到大排序,输出"1,2,3,4,5,6,7,8,9"
    </script>
  </body>
</html>
```

5. String 对象

String 对象有一个属性 length,它是字符串中的字符个数,要注意的是,即使字符串包含双字节的字符(如中文),每个字符也只算一个长度。

String 对象还提供了很多方法,在 HTML5 App 开发中常用方法如表 4-8 所示。

表 4-8 String 对象的常用方法

方法	具体描述
charAt	返回字符串对象在指定位置处的字符
charCodeAt	返回字符串对象在指定位置处字符的十进制的 ASCII 码
indexOf	要查找的字符串在字符串对象中的位置
lastIndexOf	要查找的字符串在字符串对象中的最后位置
match	字符串内检索指定的值,或找到一个或多个正则表达式的匹配
replace	在字符串中用一些字符替换另一些字符,或替换一个与正则表达式匹配的子串
search	检索字符串中指定的子字符串位置,或检索与正则表达式相匹配的子字符串位置
substr	返回一个从指定位置开始的指定长度的子字符串
substring	返回一个从指定的开始位置到结束位置的子字符串
split	以指定的字符分隔字符串
toLowerCase	返回一个将所有英文字符转换成小写字母的字符串
toUpperCase	返回一个将所有英文字符转换成大写字母的字符串

【例 4-20】 String 对象的使用示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport">
    content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>String 对象的使用</title>
  </head>
  <body>
    <script>
      var myString = "This is a sample";
      alert(myString.charAt(2));           //输出"i"
      alert(myString.charCodeAt(2));      //输出"105"
      alert(myString.indexOf("is"));      //输出"2"
      alert(myString.lastIndexOf("is"));  //输出"5"
      alert(myString.substr(10,3));
      //输出"sam",其中 10 表示位置,3 表示长度
      alert(myString.substring(5,9));
      //输出"is a",其中 5 表示开始位置,9 表示结束位置(不包括 9)
      var a = myString.split("");
      alert(a[a.length-1]);               //输出"sample"
      var newString = myString.replace("sample","Apple");
      alert(myString);                    //原字符串不变,输出"This is a sample"
      alert(newString);                   //输出"This is a Apple"
    </script>
  </body>
</html>

```



```

        alert(myString.toLowerCase());    //输出"this is a sample"
        alert(myString.toUpperCase());    //输出"THIS IS A SAMPLE"
    </script>
</body>
</html>

```

6. window 对象

window 对象表示浏览器中的一个窗口,通常在使用它时,可以直接省略 window 对象,例如“window. 方法调用”可以直接修改成“方法调用”。限于本书篇幅,只简单介绍在 HTML5 App 中会用到的一些方法。

1) 各种对话框

window 对象中的对话框有 3 种,分别是: window.alert(警告对话框)、window.confirm(确认对话框)、window.prompt(提示用户输入的对话框)。下面这个例子简单说明了几个对话框的应用。

【例 4-21】 window 对象的各对话框使用示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title>window 对象的各对话框使用</title>
  </head>
  <body>
    <script>
      if(confirm("你确定要购买这个商品?")) {
        var uname = prompt("请输入姓名:");
        alert(uname + "订购商品 1 件");
      }
    </script>
  </body>
</html>

```

三种对话框分别在 Chrome 浏览器和 Android App 中的表现形式如图 4-14 所示。

2) 间隔

JavaScript 的 window 对象提供了一个 setInterval 方法来实现间隔,它可以实现每隔一段时间自动执行代码或函数,例如在游戏中的刷帧动作,它的语法结构如下:

```
setInterval(函数名或语句,间隔的毫秒数)
```

这个方法执行后会返回一个时间间隔 ID,它是一个 number 值,这个 number 值在整个页面中是唯一的。间隔一旦打开,要考虑何时终止,否则间隔会一直执行下去,如果想取消时间间隔,语法如下:

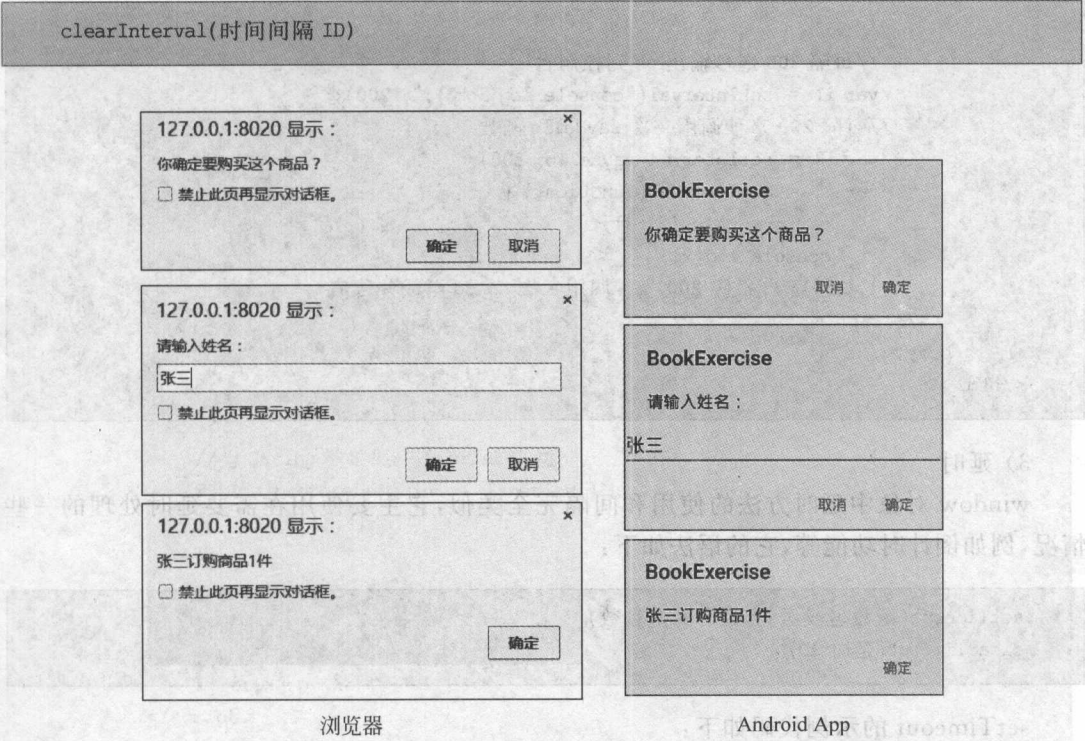


图 4-14 对话框运行效果

【例 4-22】 setInterval 的使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title>setInterval 的使用</title>
  </head>
  <body>
    <script>
      var x = 0;
      function sayHello() {
        console.log("Hello");
        x++;
        if(x >= 3) {
          clearInterval(i1);
          clearInterval(i2);
          clearInterval(i3);
        }
      }
      function doAdd(a, b) {
        return a + b;
      }
    </script>
  </body>
</html>
```

```

    }
    //每隔 200 毫秒输出 ok 到控制台
    var i1 = setInterval("console.log('ok');", 200);
    //每隔 200 毫秒调用一次 sayHello 函数
    var i2 = setInterval(sayHello, 200);
    var i3 = setInterval(function() {
        var x = doAdd(2, 3);
        console.log(x);
    }, 200); //每隔 200 毫秒调用一次 doAdd 函数并传值
</script>
</body>
</html>

```

3) 延时

window 对象中延时方法的使用和间隔完全类似,它主要使用在需要延时处理的一些情况,例如倒计时功能等,它的语法如下:

```

setTimeout(函数名或语句,延时的毫秒数)
clearTimeout(延时 ID)

```

setTimeout 的示例代码如下:

```

setTimeout("alert('游戏时间到!');",10000);
setTimeout(sayHello,3000);
setTimeout(function(){
    doAdd(2,3);
},2000);

```

4.6.3 自定义类或对象

使用内置对象的能力只是 JavaScript 语言能力的一部分,它真正强大的地方在于能创建自己专用的类和对象。这里介绍目前使用最广泛的混合的构造函数/原型方式实现自定义类。

1) prototype 属性

每个对象都有 prototype 属性,返回对象类型原型的引用,可以利用它来为类增加方法或覆盖方法,例如下面的语句:

```

var testString = "hello html5";
testString.print();

```

这段代码一旦在浏览器中运行,由于 String 对象没有定义所谓的 print 方法,浏览器的控制台会输出明显的错误信息,提示 print 方法并不存在。

```

Uncaught TypeError: testString.print is not a function

```

下面这个例子展示了如何使用 prototype 属性为 String 类添加一个 print 方法。

【例 4-23】 prototype 使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title></title>
  </head>
  <body>
    <script>
      //为 String 类扩展一个 print 方法
      String.prototype.print = function(){
        alert(this);
      };
      var testString = "hello html5";
      testString.print();
      var myString = "javascript";
      myString.print();
    </script>
  </body>
</html>
```

在上面这个例子中,为 String 类扩展了一个 print 方法,由于 testString 和 myString 都是 String 类的实例,所以它们都自动拥有了 test 方法。prototype 属性不光能为自定义类扩展属性和方法,对于 JavaScript 内置的类也同样有效,这是其他编程语言没有的特性。

2) 混合的构造器/原型方式实现自定义类

这种方式是比较简单的一种方式,即用构造器定义类的所有属性,而用原型方式定义类的方法。所有的函数只创建一次,而每个对象都具有自己的对象属性实例。

【例 4-24】 定义一个 Student 类并生成对象,示例代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title>定义类并生成对象</title>
  </head>
  <body>
    <script>
      //Student 类的构造函数
      function Student(name, age) {
        this.name = name;
```



```

        this.age = age;
    }
    //为 Student 类添加一个 showStuInfo 方法
    Student.prototype.showStuInfo = function() {
        alert("姓名:" + this.name + ",年龄:" + this.age);
    }
    //生成对象并调用方法
    var ostu = new Student("张三", 22);
    ostu.showStuInfo();
</script>
</body>
</html>

```

上面这个例子中,JavaScript 没有关键字 `class`,定义类的构造器时就是采用 `function`。Student 类所有的属性都在其构造器中定义,而方法是在 `function` 构造器之外,使用 `prototype` 原型方式附加上去。

3) this 关键字

JavaScript 中要掌握的重要概念之一就是关键字 `this` 的用法,它总是指向调用该方法的对象。例如,例 4-24 中的 Student 类中使用了一个 `function` 来定义 Student 类的构造器,但是这个 `function` 到底是函数是类构造器呢?从下面的例子可以看出区别。

【例 4-25】 `this` 关键字的使用,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport">
    content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title>this 关键字的使用</title>
  </head>
  <body>
    <script>
      function Student(name, age) {
        alert(this);
        this.name = name;
        this.age = age;
      }
      Student("张三", 22);           //作为简单函数使用
      alert(window.name + ", " + window.age);
      var ostu = new Student("张三", 22); //作为类的构造器使用
      alert(ostu.name + ", " + ostu.age);
    </script>
  </body>
</html>

```

这个例子演示了 function 作为一般函数和作为类的构造器的区别,当有 new 关键字时,function 一定是类的构造器。当 Student 作为一般函数使用时,this 指向 window 对象,而作为构造器时,this 指向 Student 对象。在构造器中,类的属性前面一定记得加 this 关键字。

4.7 JavaScript 处理 JSON

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式,采用完全独立于语言的文本格式,是理想的数据交换格式。同时,JSON 是 JavaScript 原生格式,这意味着在 JavaScript 中处理 JSON 数据不需要任何特殊的 API 或工具包。目前,在各种 App 与服务器的交互中,JSON 已经成为流行的数据格式。

4.7.1 JSON 格式结构简介

JSON 对象建构于两种结构:

(1) 单一对象,一个对象以“{”开始,以“}”结束,对象有多个属性,属性必须使用双引号括起来,属性的值以“:”赋值,属性之间使用逗号“,”间隔。下面是一个最简单的 JSON 对象示例:

```
var student = {"name": "张三", "age": 22};  
alert("姓名:" + student.name + ", 年龄:" + student.age);
```

从这个例子看出,JSON 数据格式结构简单,读取相应属性时只需一个点“.”就可以获取,在程序中解析数据是非常方便的。

(2) 对象集合,当有多个对象时,采用“[”开始,“]”结束,中间放多个对象,对象之间以逗号“,”间隔,对象的结构必须完全一致。下面是包含 3 个对象的 JSON 对象集合:

```
var students = [{"name": "张三", "age": 22},  
                {"name": "李四", "age": 21}, {"name": "王五", "age": 20}];  
alert(students[1].name);
```

和数组类似,访问其中的某个对象是以索引号访问。

4.7.2 JSON 序列化与反序列化

在目前的各种 App 开发中,经常涉及手机与服务器的交互,例如向服务器提交数据,或从服务器取回数据并解析,这就要求必须掌握将 JSON 对象序列化成字符串,将 JSON 格式字符串反序列化成 JSON 对象。

1. 序列化成字符串

JSON 已经是 JavaScript 标准的一部分。目前,主流的浏览器引擎对 JSON 支持都非常完美。可以使用 JSON.stringify 方法来实现序列化,例如:

```
var student = {"name": "张三", "age": 22};
alert(student.toString()); //输出"object"
var stuString = JSON.stringify(student);
alert(stuString); //输出{"name": "张三", "age": 22}
alert(typeof(stuString)); //输出"string"
```

2. 反序列化对象

反序列化是序列化的反向动作,可以使用 JSON.parse 方法,它主要是将完全符合 JSON 格式规则的字符串还原成 JavaScript 对象,如果字符格式不正确,该方法会报错,例如:

```
var studentString = '{"name": "张三", "age": 22}';
alert(typeof(studentString)); //输出"string"
var ostu = JSON.parse(studentString); //反序化成对象
alert(ostu.name); //输出"张三"
var xString = "1242,234";
alert(JSON.parse(xString)); //格式不正确,报错到控制台
```

3. 使用 Chrome 帮助解析数据

上面举例使用的 JSON 数据格式都相对比较简单,但实际 App 开发过程中,需要解析的 JSON 数据格式有可能非常复杂,例如集合中嵌套有对象,对象的某个属性又是一个集合等。这里用一个例子来说明如何使用 Chrome 的开发者工具,帮助程序员解析数据。

【例 4-26】 使用 Chrome 帮助解析复杂的 JSON 格式数据,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title> Chrome 帮助解析复杂的 JSON 格式数据</title>
  </head>
  <body>
    <script>
      var dataString = '{"teachers":[{"name": "黄波", "course": "网页高级设计"}, {"name": "黄平", "course": "Java 程序设计"}, {"name": "张小华", "course": "JavaScript 程序设计"}], "students":[{"name": "张义", "age": 20}, {"name": "李四", "age": 21}, {"name": "王五", "age": 19}]}';
      var odata = JSON.parse(dataString);
    </script>
  </body>
</html>
```

这个 JSON 格式字符串 dataString 相对复杂一些,包含了 3 个教师的数据和 3 个学生的数据。例如,要取出“李四”的年龄,用肉眼分辨不太可能实现,也容易出错,那么该如何实

现呢？

使用 Chrome 强大的“开发者工具”，断点调试，并将变量 odata 添加到监视窗口后，让 odata 获得反序列化后的值，这时 odata 已成为 JSON 对象，在监视窗口中将 odata 展开，如图 4-15 所示。

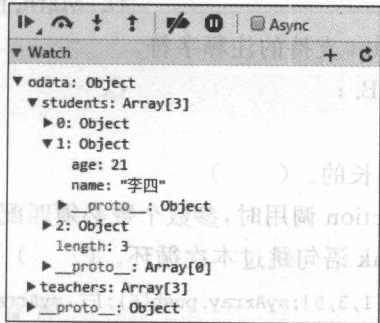


图 4-15 JSON 对象展开

找到要获取的“李四”数据位置，从 odata 结点开始，按树形的结构依次完成结点访问代码：odata.students[1].age，这就可以得到需要的值了，完整代码如下：

```
var dataString = '{"teachers":[{"name":"黄波","course":"网页高级设计"}, {"name":"黄平","course":"Java 程序设计"}, {"name":"张小华","course":"JavaScript 程序设计"}], "students": [{"name":"张义","age":20}, {"name":"李四","age":21}, {"name":"王五","age":19}]}';
var odata = JSON.parse(dataString);
alert(odata.students[1].age);
```

小结

本章主要讲解了 JavaScript 的编程基础，介绍了 JavaScript 语言的特点，讲解了 JavaScript 在 HTML5 页面中的使用，详细介绍了基础语法，强调了调试技巧，然后讲解了函数、各种内置对象，以及如何实现自定义类和对象，对目前流行的 JSON 数据格式以及序列化和反序列化也作了详细的讲解。本章所讲解的内容在实际开发中非常重要，因此要求必须熟练掌握，为后面的学习做好铺垫。

习题

一、选择题

- 1. 我们可以在下列()HTML 标签中放置 JavaScript 代码。
A. javascript B. script C. js D. ecmascript
- 2. 引用名为"xxx.js"的外部脚本的正确语法是()。
A. <script src = "xxx.js"> B. <script href = "xxx.js">
C. <script name = "xxx.js"> D. <script link = "xxx.js">

3. 下面()选项是编写当 i 等于 5 时执行一些语句的条件语句。

- A. if(i = 5) B. if i = 5 then C. if i = 5 D. if i = 5 then

4. 下面()选项是 7.25 四舍五入最为接近的整数。

- A. Math.round(7.25) B. Math.rnd(7.25)
C. Math.floor(7.25) D. Math.max(7.25)

5. 下面()是 JavaScript 支持的注释字符。

- A. // B. ; C. — D. &&

二、判断题

- JavaScript 的数组是定长的。()
- JavaScript 的函数 function 调用时,参数个数必须匹配。()
- JavaScript 中使用 break 语句跳过本次循环。()
- 执行数组 var myArray = [1,3,5];myArray.push(6);后,myArray = [6,1,3,5]。()
- JavaScript 如果有语法错误,IDE 和浏览器都会自动提示。()

三、填空题

- JavaScript 中声明变量使用的关键字是_____。
- JavaScript 中的恒等运算符为_____,用于比较两个运算数的值相等,而且数据类型也相同。

3. JavaScript 使用关键字_____创建自定义类。

4. JavaScript 使用关键字_____实现间隔,关键字_____实现延时。

四、简答题

- 试述 JavaScript 中全局变量与局部变量的作用域。
- 试述 JavaScript 的 this 关键字的特性。

五、编程题

- 试实现一个 for 循环并用 Chrome 调试。
- 试自定义 JSON 对象和字符串,并练习序列化与反序列化。

第 5 章

CHAPTER 5

JavaScript 交互编程

【例 5-1】 document 对象的各属性使用示例,代码如下:

学习目标

- 了解 DOM 概念。
- 掌握 document 对象的使用。
- 掌握使用 DOM 查找节点的各种方法。
- 掌握使用 DOM 进行 HTML 元素属性控制、HTML 内容控制。
- 掌握使用 DOM 进行创建和操作 HTML 元素节点。
- 掌握使用 DOM 进行样式编程。
- 了解事件的概念,掌握常用的一些事件以及事件的监听方法。

DOM 操作与事件是 JavaScript 最核心的组成部分之一,它们赋予了页面无限的想象空间,在 HTML5 App 中就是依靠它们实现交互的。本章主要讲解在 HTML5 App 开发中必须掌握的一些 DOM 编程基础以及事件的使用,以便于实现高效和便捷的页面交互。

5.1 DOM 介绍

DOM(Document Object Model,文档对象模型)是 HTML 和 XML 的应用程序接口(API)。DOM 将整个页面规划成由节点层级构成的文档。例如下面这个 HTML 页面:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>测试页面</title>
  </head>
  <body>
    <p>Hello HTML5 </p>
  </body>
</html>
```

这段 HTML 代码可以用 DOM 绘制成一个节点层次图,如图 5-1 所示。

DOM 通过创建树来表示 HTML 文档,从而使开发者对文档的内容和结构具有很强的控制力,可以使用 DOM API 对这棵树的节点作各种变化:增加节点、删除节点、查找节点、修改节点等,DOM 技术还使得用户页面可以动态地变化,如可以动态地显示或隐藏一个元

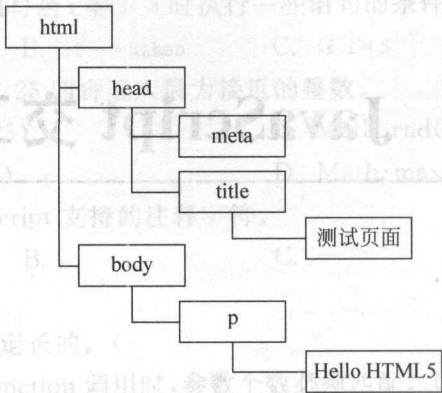


图 5-1 HTML 节点层次

素、改变它们的属性、增加一个元素等，大大地增强了页面的交互性。

5.2 使用 DOM

在 HTML5 App 开发的过程中,JavaScript 极为重要的一个功能就是 DOM 对象的操作,本节将讨论各种 DOM 操作,以便于实现高效率 and 便捷的页面交互。

5.2.1 document 对象

在浏览器引擎中,与用户进行数据交换都是通过客户端的 JavaScript 代码来实现的,而完成这些交互工作大多数是由 document 对象及其部件进行的,因此 document 对象是一个比较重要的对象。document 对象是文档的根节点,window. document 属性就指向这个对象。也就是说,只要浏览器开始载入 HTML 文档,这个对象就开始存在了,就可以直接调用。

表 5-1 列出了 document 对象的常用属性。

表 5-1 document 对象的常用属性

属 性	具体描述
alinkColor	表示激活链接(焦点在此链接上)的颜色
bgColor	表示页面背景色
body	表示< body>节点
charset	表示页面的字符集
doctype	表示文档类型节点,也就是<!DOCTYPE html>节点
documentElement	表示< html>节点
fgColor	表示前景色(文本颜色)
forms	表示页面上的所有 form 元素
head	表示< head>节点
images	表示页面上的所有 img 元素
lastModified	表示最终修改的日期
linkColor	表示未单击过的链接颜色

续表

属 性	具 体 描 述
links	表示页面上的所有 a 元素
scripts	表示页面上的所有 script 元素
styleSheets	表示页面上的所有 link 或 style 元素
title	表示 title 的内容
URL	表示当前文档的 URL
vlinkColor	表示已单击过的链接颜色

【例 5-1】 document 对象的各属性使用示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf - 8">
    <metaname = "viewport"
content = "initial - scale = 1.0,maximum - scale = 1.0, user - scalable = no" />
    <title>document 对象测试</title>
    <style>
      img {
        width: 120px;
        height: 80px;
      }
    </style>
    <link href = "1.css" rel = "stylesheet"/>
  </head>
  <body>
    <img src = "../img/baidu.png" />
    <img src = "../img/163.png" /><br />
    <a href = "http://www.baidu.com">百度</a>
    <a href = "http://www.163.com">网易</a>
    <p>这只是一段文字测试</p>
    <script src = "1.js"></script>
    <script>
      document.alinkColor = "yellow";
      document.vlinkColor = "brown";
      document.bgColor = "bisque";
      document.fgColor = "crimson";
      alert(document.doctype);           //输出[object DocmentType]
      alert(document.body);              //输出[object HTMLBodyElement]
      alert(document.head);              //输出[object HTMLHeadElement]
      alert(document.title);              //输出"document 对象测试"
      alert("页面上有" + document.scripts.length + "个 script 标签");
                                           //输出"页面上有 2 个 script 标签"
      alert(document.documentElement);    //输出[object HTMLHtmlElement]
      alert("页面上有" + document.links.length + "个超链接");
                                           //输出"页面上 2 个超链接"
    </script>
  </body>
</html>

```

```

        alert("页面上有" + document.images.length + "张图片");
                                //输出"页面上有 2 张图片"
        alert("页面上有" + document.styleSheets.length + "处样式");
                                //输出"页面上有 2 处样式"
        alert(document.lastModified); //输出页面最后修改的日期
        alert(document.URL);          //输出当前页面的 URL 地址
    </script>
</body>
</html>

```

5.2.2 查找节点

在 HTML5 程序开发中,经常要修改某个 HTML 元素的样式、内容等,如何获取相应的元素,是首先要解决的问题,DOM 中提供了一些方法来方便快捷地访问指定的 HTML 元素节点,以下分别讲解。

1. getElementsByTagName 方法

这个方法用来返回一个页面上所有包含 tagName(标签名)等于某个指定值的元素节点对象集合。当得到相应的节点集合以后,就可以使用方括号来访问其中某个子节点。

【例 5-2】 getElementsByTagName 使用示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>getElementsByTagName 使用</title>
    <style>
      img{
        width: 200px;
        height: 100px;
      }
    </style>
  </head>
  <body>
    <br />
    <input type="text" value="hello world" /><br />
    <input type="password" value="123456" />
    <script>
      var oImg = document.getElementsByTagName("img");
      alert(oImg[0].tagName); //输出"IMG"
      oImg[0].src = "../../img/163.png";
      var oInput = document.getElementsByTagName("input");
      alert(oInput[0].value); //输出"hello world"
      oInput[0].value = "Hello HTML5";
    </script>
  </body>
</html>

```



```
</body>
</html>
```

在这个例子中,从节点对象中取得某个标签节点对象,意味着我们可以对标签相应的属性进行读取或设置。按照第 4 章讲解的调试断点方法,可以看到节点对象的属性和方法,如图 5-2 所示为 oImg[0]节点对象属性的部分截图。

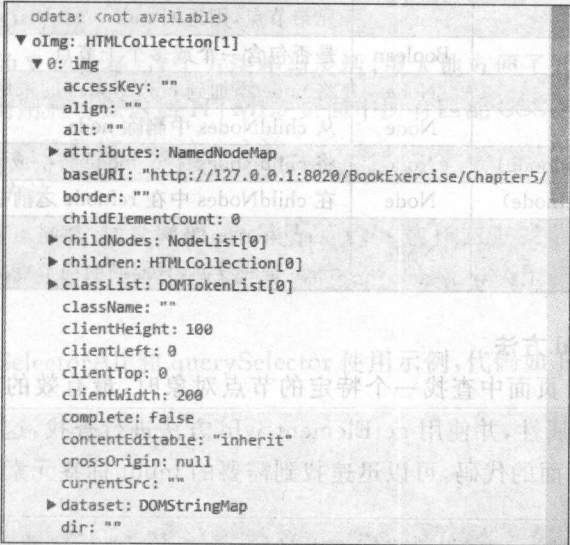


图 5-2 节点对象属性的部分截图

在 HTML DOM 中,每个部分都是节点:

- 文档本身是文档节点;
- 所有 HTML 元素是元素节点;
- 所有 HTML 属性是属性节点;
- HTML 元素内的文本是文本节点;
- 注释是注释节点。

节点对象在 DOM 中定义为 Node 对象,Node 对象定义了一些属性和方法,表 5-2 中列出了这些属性和方法。

表 5-2 Node 对象的常用属性和方法

属性/方法	返回类型	具体描述
innerHTML	String	表示当前节点的内部标签
innerText	String	表示当前节点的文字内容
length	Number	返回 NodeList 中的节点数
nodeName	String	节点名称,根据节点的类型而定义
nodeValue	String	节点的值,根据节点的类型而定义
nodeType	Number	节点的类型常量值之一
firstChild	Node	指向在 childNodes 节点集合中的第一个节点
lastChild	Node	指向在 childNodes 节点集合中的最后一个节点

续表

属性/方法	返回类型	具体描述
parentNode	Node	指向所在节点的父节点
childNodes	NodeList	所有子节点的集合
previousSibling	Node	指向前一个兄弟节点,如果当前节点本身就是第一个兄弟节点,则返回 null
nextSibling	Node	指向后一个兄弟节点,如果当前节点本身就是最后一个兄弟节点,则返回 null
hasChildNodes()	Boolean	是否包含一个或多个子节点
AppendChild(node)	Node	将 node 添加到 childNodes 的末尾
removeChild(node)	Node	从 childNodes 中删除 node
replaceChild(newnode,oldnode)	Node	将 childNodes 中 oldnode 替换成 newnode
insertBefore(newnode,refnode)	Node	在 childNodes 中在 refnode 之前插入 newnode
cloneNode(deep)	Node	deep 为 true 是深复制,复制当前节点以及子节点,为 false 是浅复制,只复制当前节点

2. getElementById 方法

当需要在 HTML 页面中查找一个特定的节点对象时,最有效的方法就是为该节点的标签元素添加一个 id 属性,并使用 getElementById 方法进行查找,这个方法会返回唯一的节点对象,例如通过下面的代码,可以迅速找到需要的 input 标签元素:

```
<input type = "text" value = "hello html5" id = "myTest"/>
<script>
    var oInput = document.getElementById("myTest");
    alert(oInput.value);
</script>
```

3. getElementsByClassName 方法

当需要在 HTML 页面中查找多个对象时,可以为这些对象指定相同的 class 属性,并使用 getElementsByClassName 方法进行查找,这个方法会返回所有 class 属性为指定值的节点对象集合。

【例 5-3】 getElementsByClassName 使用示例,代码如下:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8">
        <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
        <title>getElementsByClassName 使用</title>
    </head>
    <body>
        <div class = "example">第 1 个 div</div>
        <div>第 2 个 div</div>
```

```

<p class = "example">第 1 个 p</p>
<script>
    var elems = document.getElementsByClassName("example");
    alert(elems[1].tagName); //输出 "p"
</script>
</body>
</html>

```

4. querySelectorAll 方法

作为查找 DOM 的又一途径,这个方法相当灵活,极大地方便了开发者。它可以接受一个 CSS 选择器参数,调用后可以返回 HTML5 页面中所有匹配 CSS 选择器的元素节点对象集合,目前所有的主流浏览器都支持这一方法。

5. querySelector 方法

和方法 querySelectorAll 完全类似,也是使用 CSS 选择器查找节点,不同的是,这个方法只返回匹配选择器的第 1 个元素节点对象,而 querySelectorAll 返回的是所有匹配的元素节点对象集合。

【例 5-4】 querySelectorAll 和 querySelector 使用示例,代码如下:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf - 8">
        <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
        <title>querySelectorAll 和 querySelector 的使用</title>
        <style>
            div {
                border: 1px dotted #ccc;
                padding: 8px;
                width: 200px;
                margin - bottom: 30px;
            }
        </style>
    </head>
    <body>
        <div id = "test">
            我是 id 为 test 的 div
        </div>
        <div class = "mytest">
            <p>我是 div 里的 p 标签</p>
        </div>
        <script>
            var oDiv1 = document.querySelector("#test");
            alert(oDiv1.innerText);           //输出"我是 id 为 test 的 div"
            var oDiv2 = document.querySelectorAll("#test");
            alert(oDiv2[0].innerText);        //输出"我是 id 为 test 的 div"
        </script>
    </body>
</html>

```



```

        var oP1 = document.querySelector("div.mytest>p");
        alert(oP1.innerText);           //输出"我是 div 里的 p 标签"
        var oP2 = document.querySelectorAll("div.mytest>p");
        alert(oP2[0].innerText);       //输出"我是 div 里的 p 标签"
    </script>
</body>
</html>

```



查找 HTML 元素节点时，应确保它已在 DOM 树上构建，否则会出现找不到的情况。

例如下面这种情况，就未能成功查找到按钮对象，必须把 JavaScript 代码放在 HTML 代码之后：

```

<script>
    var oInput = document.getElementById("myButton");
    alert(oInput);           //输出 null
</script>
<input type="button" value="测试" id="myButton"/>

```



为了提高程序性能，一定要避免重复的 DOM 查找，例如下面这段代码就会造成性能问题：

```

<script>
    for(var i = 0; i < 10; i++){
        var oDiv = document.getElementById("mydiv");
        ...
    }
</script>

```

正确的方式是在循环体之外声明一个变量用来存储查找出来的 HTML 元素节点对象，修改后的代码如下：

```

<script>
    var oDiv = document.getElementById("mydiv");
    for(var i = 0; i < 10; i++)
    {
        ...
    }
</script>

```

5.2.3 处理属性

对于 HTML 元素节点，DOM 提供了三个方法来处理其属性，这些方法相当有用：

- `getAttribute(name)`: 获取某个属性的值;
- `setAttribute(name,newvalue)`: 设置某个属性的值;
- `removeAttribute(name)`: 移除某个属性。

【例 5-5】 DOM 控制 HTML 元素属性示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title>HTML 元素对象属性的处理</title>
  </head>
  <body>
    <input type = "button" value = "测试" id = "mybutton"
data-test = "1234" />
    <script>
      var oInput = document.getElementById("mybutton");
      alert(oInput.getAttribute("type"));    //输出"button"
      oInput.setAttribute("type", "text");  //将按钮切换成输入框
      //设置自定义属性
      oInput.setAttribute("data-myattr", "just a test");
      //读取自定义属性,输出"just a test"
      alert(oInput.getAttribute("data-myattr"));
      oInput.removeAttribute("id");        //输出"id"属性
    </script>
  </body>
</html>
```

这里介绍自定义属性,HTML5 标准中规定自定义属性需要添加前缀 `data-`,目的是提供与页面渲染无关的信息,运行这个例子后会发现, `input` 标签添加了 `data-test` 属性后,对于界面的显示并无任何影响。使用自定义属性可以很方便地存储页面或应用程序的私有自定义数据,目前所有主流浏览器都支持 `data-*` 属性。



HTML 标签元素中只有标准属性才会以属性的形式添加到 DOM 对象中,DOM 对象只能访问这些标准属性,而 `getAttribute` 方法既可以访问标准属性也可以访问自定义属性。

5.2.4 读取和设置内容

在 HTML5 App 开发中,经常涉及读取或设置 HTML 标签元素内部所包含的文字或子 HTML 标签,DOM 提供了以下属性以供使用。

1. `innerText` 属性

DOM 中通过 `innerText` 属性可以操作元素中包含的所有文本内容,无论文本位于子文档树中的什么位置。在通过 `innerText` 读取值时,它会按照由浅入深的顺序,将子文档树中

所有文本拼接起来。以下面的 HTML 代码为例。

【例 5-6】 innerText 属性使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title> innerText 使用示例</title>
  </head>
  <body>
    <div id = "content">
      <p>This is a <strong> paragraph</strong>
      with a list following it.</p>
      <ul>
        <li> item1 </li>
        <li> item2 </li>
        <li> item3 </li>
        <li> item4 </li>
      </ul>
    </div>
    <script>
      var oDiv = document.getElementById("content");
      alert(oDiv.innerText);
      oDiv.innerText = "hello html5";
    </script>
  </body>
</html>
```

对于这个例子中的 div 而言,其 innerText 会返回下列字符串:

This is a paragraph with a list following it.

item1
item2
item3
item4

设置 div 的 innerText,它的内容变成了:

```
<div id = "content"> hello html5 </div>
```



使用 innerText 或 innerHTML 为 HTML 元素对象设置内容时,会先将对象开始标签和结束标签之间的内容全清空。

2. innerHTML 属性

几乎所有的 DOM 对象都有 innerHTML 属性,它是一个字符串,用来设置或获取位于

HTML 标签对象起始和结束标签内的 HTML 代码。

【例 5-7】 innerHTML 属性使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title> innerHTML 使用示例</title>
  </head>
  <body>
    <div id = "content">
      <p>This is a <strong>paragraph</strong> with a list following it.</p>
      <ul>
        <li> item1 </li>
        <li> item2 </li>
        <li> item3 </li>
        <li> item4 </li>
      </ul>
    </div>
    <script>
      var oDiv = document.getElementById("content");
      alert(oDiv.innerHTML);
      oDiv.innerHTML = '<img src = "../img/baidu.png" />';
    </script>
  </body>
</html>
```

对于这个例子中的 div 而言,其 innerHTML 会返回下列 HTML 字符串:

```
<p>This is a <strong>paragraph</strong> with a list following it.</p>
<ul>
  <li> item1 </li>
  <li> item2 </li>
  <li> item3 </li>
  <li> item4 </li>
</ul>
```

设置 div 的 innerHTML 后,它的内容会变成一张图片显示:

```
<div id = "content">
  <img src = "../img/baidu.png" />
</div>
```



DOM 操作结束后,如果单击鼠标右键,单击“查看网页源代码”后,会发现源代码没有任何变化。要查看变化后的标签,必须使用 Chrome 浏

浏览器的“开发者工具”进行查看，这是用 HTML5 App 开发调试必须掌握的技能。

5.2.5 创建和操作节点

前面已经介绍了如何利用 DOM 查找 HTML 元素节点，不过这只是 DOM 所能实现功能的一小部分，DOM 还可以添加、删除、替换(或其他操作)节点。正是这些功能才使得 DOM 具有真正意义上的动态性和交互性。

1. 创建新节点

对于一个好的页面来说，为了让用户体验做到极致，动态创建页面节点是必不可少的。DOM 中有一些方法可以用于创建不同类型的节点，最常用到的几个方法见表 5-3。

表 5-3 创建节点的常用方法

方 法	具体描述
createTextNode(text)	创建包含文本 text 的文本节点
createElement(tagName)	创建标签为 tagName 的 HTML 元素节点
createDocumentFragment()	创建文档碎片节点

2. createElement、createTextNode、AppendChild

以如下的 HTML5 页面为例：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title></title>
    <style>
      #mydiv{
        width: 150px;
        height: 150px;
        background - color: red;
        margin: 0 auto;
        text - align:center;
      }
    </style>
  </head>
  <body>

  </body>
</html>
```

现在想使用 DOM 操作来将以下 HTML 代码添加到页面中：

```
<div id = "mydiv">
  <p>HELLO HTML5 </p>
</div>
```

这里可以使用 `createElement` 方法和 `createTextNode` 方法来实现,下面是实现步骤:
首先,创建 `div` 元素,并设置其 `id` 属性值为“mydiv”:

```
var oDiv = document.createElement("div");
oDiv.setAttribute("id", "mydiv");
```

第二步,创建 `p` 元素:

```
var oP = document.createElement("p");
```

第三步,创建文本节点:

```
var oText = document.createTextNode("HELLO HTML5");
```

最后,使用 `AppendChild` 方法依次把各子节点添加到相应节点的 `childNodes` 的尾部:

```
oP.AppendChild(oText);
oDiv.AppendChild(oP);
document.body.AppendChild(oDiv);
```

【例 5-8】 动态创建节点使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>动态创建节点使用示例</title>
    <style>
      #mydiv {
        width: 150px;
        height: 150px;
        background-color: red;
        margin: 0 auto;
        text-align: center;
      }
    </style>
  </head>
  <body>
    <script>
      var oDiv = document.createElement("div");
      oDiv.setAttribute("id", "mydiv");
      var oP = document.createElement("p");
      var oText = document.createTextNode("HELLO HTML5");
      oP.AppendChild(oText);
```



```

        oDiv.appendChild(oP);
        document.body.appendChild(oDiv);
    </script>
</body>
</html>

```

运行后的界面和页面的 HTML 动态变化如图 5-3 所示。

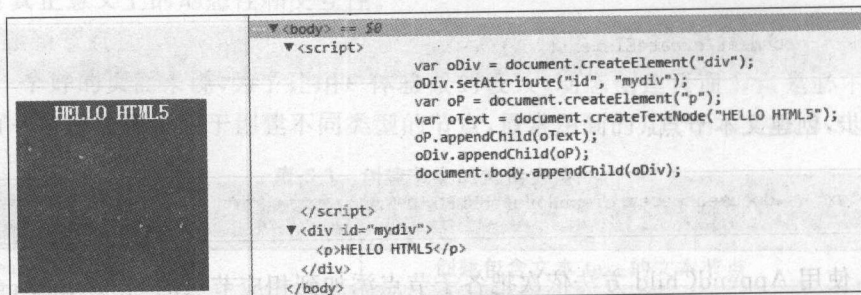


图 5-3 动态创建节点效果



所有的 DOM 操作必须在页面完全载入后才能进行。当页面正在载入时，要向 DOM 插入相关节点是不可能的，因为 DOM 树还没有构建完成。

3. removeChild

既然可以添加节点，当然也可以删除节点，这就是 removeChild 方法所能完成的。这个方法接受一个参数，代表要删除的节点对象，返回值也是这个节点对象，删除时要尽量使用节点的 parentNode 特性来确保能访问到它真正的父节点。如图 5-4 所示的例子中，需要移除其中红色的小 div 节点。

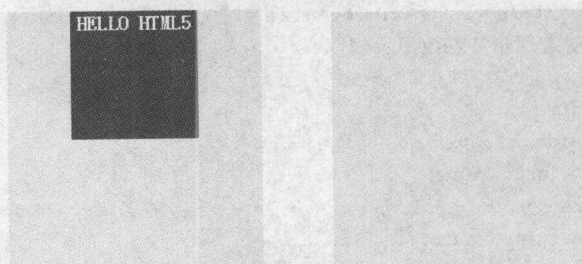


图 5-4 节点移除前后效果

【例 5-9】 动态移除节点使用示例，代码如下：

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"

```

```

content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
<title>动态移除节点使用示例</title>
<style>
  div{
    margin: 0 auto;
    text-align: center;
  }
  #parent{
    width: 200px;
    height: 200px;
    background-color: yellow;
  }
  #child{
    width: 100px;
    height: 100px;
    background-color: red;
  }
</style>
</head>
<body>
  <div id = "parent">
    <div id = "child">
      HELLO HTML5
    </div>
  </div>
  <script>
    var oDiv = document.getElementById("child");
    oDiv.parentNode.removeChild(oDiv);
  </script>
</body>
</html>

```

这个页面加载后,在看到它之前,红色的小 div 已被自动移除。

4. replaceChild

如果想将节点替换成新的节点,则需要使用 replaceChild 方法。replaceChild 方法有两个参数——被添加的节点对象和被替换的节点对象。例如图 5-5 中,需要将第一个节点替换。

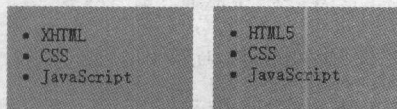


图 5-5 节点替换前后效果

【例 5-10】 动态替换节点使用示例,代码如下:

```

<!DOCTYPE html>
<html>

```

```

<head>
  <meta charset = "UTF - 8">
  <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
  <title>动态替换节点使用示例</title>
  <style>
    body{
      background - color: #ccc;
    }
  </style>
</head>
<body>
  <ul id = "dataList">
    <li>XHTML</li>
    <li>CSS</li>
    <li>JavaScript</li>
  </ul>
  <script>
    var dList = document.getElementById("dataList");
    var oLi = document.querySelector("#dataList li:first-child");
    var nLi = document.createElement("li");
    nLi.innerText = "HTML5";
    dList.replaceChild(nLi, oLi);
  </script>
</body>
</html>

```

5. insertBefore

当向页面中添加节点时,如果想让新节点出现在某个节点之前,就使用 insertBefore 方法。这个方法接受两个参数——要添加的节点对象和目标节点对象。例如图 5-6 中,需要在列表项 CSS 之前插入列表项 PhotoShop。

- 
- HTML5
 - CSS
 - JavaScript
- HTML5
 - PhotoShop
 - CSS
 - JavaScript

图 5-6 节点插入前后效果

【例 5-11】 动态插入节点使用示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title>动态插入节点使用示例</title>

```



```

<style>
  body{
    background-color: #ccc;
  }
</style>
</head>
<body>
  <ul id="dataList">
    <li>HTML5 </li>
    <li>CSS </li>
    <li>JavaScript </li>
  </ul>
  <script>
    var dList = document.getElementById("dataList");
    var oLi = document.querySelector("#dataList li:nth-of-type(2)");
    var nLi = document.createElement("li");
    nLi.innerText = "PhotoShop";
    dList.insertBefore(nLi, oLi);
  </script>
</body>
</html>

```

6. createDocumentFragment

在 HTML5 App 开发中,经常会遇到根据服务器返回的数据,在某个节点处生成多个列表项的场景,通常这部分可以根据数据的条数,使用 createElement 循环生成对应的节点对象后,附加到相应的父节点,但 DOM 修改会导致页面重绘、重新排版。重新排版会阻塞用户的操作,同时,如果频繁重排,CPU 使用率也会猛涨,App 的性能会受到严重影响。所以,为了得到更高的性能,一般使用 createDocumentFragment 创建文档碎片,把所有的新节点附加其上,然后把文档碎片一次性添加到指定的节点上。

【例 5-12】 createDocumentFragment 使用示例,向一个 ul 添加 10000 条列表项,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>createDocumentFragment 使用示例</title>
  </head>
  <body>
    <ul id="dataList">
    </ul>
    <script>
      var dList = document.getElementById("dataList");

```

```

var fragment = document.createDocumentFragment();
for(var i = 0; i < 10000; i++)
{
    var oLi = document.createElement("li");
    oLi.innerText = "Item" + (i + 1);
    fragment.appendChild(oLi);
}
dList.appendChild(fragment);
</script>
</body>
</html>

```

7. cloneNode

cloneNode 这个方法主要用来实现对节点对象的复制,并返回复制的节点对象。它有一个输入参数,类型是 Boolean,默认为 false,表示浅复制,如果是 true,表示深复制。

【例 5-13】 cloneNode 使用示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <title>cloneNode 方法使用示例</title>
    <style>
      .mylist{
        width: 200px;
        height: 100px;
        background-color: yellow;
      }
    </style>
  </head>
  <body>
    <div id = "content">
      <ul class = "mylist">
        <li>HTML5 </li>
        <li>CSS3 </li>
        <li>JavaScript </li>
      </ul>
    </div>
    <script>
      var oDiv = document.getElementById("content");
      var oUl = document.getElementsByClassName("mylist")[0];
      //实现深复制
      oDiv.appendChild(oUl.cloneNode(true));
      //实现浅复制
      oDiv.appendChild(oUl.cloneNode(false));
    </script>
  </body>
</html>

```



```

        .mynewstyle{
            border: 2px solid black;
            background-color: lightskyblue;
        }
    </style>
</head>
<body>
    <div id="mydiv" class="mystyle">
    </div>
    <span id="desc"></span>
    <script>
        var oDiv = document.getElementById("mydiv");
        var oSpan = document.getElementById("desc");
        oSpan.innerText = "div 的样式为:" + oDiv.className;
        setTimeout(function(){
            oDiv.className = "mynewstyle";
            oSpan.innerText = "div 的样式为:" + oDiv.className;
            setTimeout(function(){
                oDiv.className = "";
                oSpan.innerText = "移除 div 的所有样式";
            },2000);
        },2000);
    </script>
</body>
</html>

```

程序运行后,先在页面上显示一个黄色的 div,2s 后自动变成一个带边框的淡蓝色 div,再过 2s 颜色和边框自动消失,效果如图 5-8 所示。

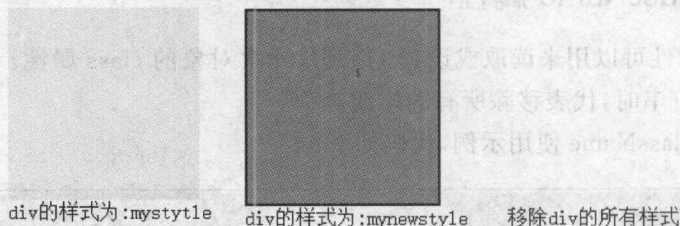


图 5-8 className 修改样式和移除样式

5.3.2 classList 对象

className 属性使用比较简单,但如果 HTML 元素对象的 class 属性值中有多个样式类应用时,对其样式分别进行控制就不太方便,例如:

```
<div id="mydiv" class="style1 style2 style3"></div>
```

为了解决这个问题,在 HTML5 API 里,页面 DOM 里的每个节点上都有一个 classList 对象,提供了如表 5-4 所示的方法新增、删除、修改节点上的样式类。

表 5-4 classList 对象方法和属性

方 法	具 体 描 述
length	返回 HTML 元素对象 class 属性中样式类的个数
add(className)	给 HTML 元素对象的 class 属性添加一个样式类
remove(className)	从 HTML 元素对象的 class 属性中删除一个指定的样式类
toggle(className)	若 HTML 元素对象的 class 属性有指定的样式类,则执行 remove 操作,没有则执行 add 操作
contains(className)	检测 HTML 元素对象的 class 属性中是否包含指定的样式类

【例 5-15】 classList 对象使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0,maximum - scale = 1.0, user - scalable = no" />
    <title>classList 对象使用示例</title>
    <style>
      div{
        width: 150px;
        height: 150px;
      }
      .mystyle1{
        background - color: yellow;
      }
      .mystyle2{
        border: 2px solid black;
      }
      .mystyle3{
        border - radius: 75px;
      }
    </style>
  </head>
  <body>
    <div id = "mydiv" class = "mystyle1 mystyle2">
    </div>
    <span id = "desc1"></span><br />
    <span id = "desc2"></span>
    <script>
      var oDiv = document.getElementById("mydiv");
      var oSpan1 = document.getElementById("desc1");
      var oSpan2 = document.getElementById("desc2");
      oSpan1.innerHTML = "div 的样式类数目: " + oDiv.classList.length;
      oSpan2.innerHTML = "包含 style2 样式类?"
        + oDiv.classList.contains("mystyle2");
      setTimeout(function(){
        oDiv.classList.add("mystyle3");
        oDiv.classList.remove("mystyle2");
      },3000);
```

```
setInterval(function(){
    oDiv.classList.toggle("mystyle1");
},1000);
</script>
</body>
</html>
```

程序运行后,先在页面上显示一个黄色的带边框的矩形 div,1s 后自动变成一个无边框的圆形 div,这个圆形 div 会每隔 1s 闪烁出现一次,效果如图 5-9 所示。

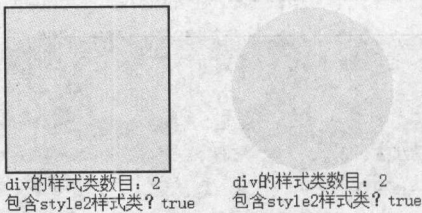


图 5-9 classList 对象动态添加、移除样式



className 属性和 classList 对象在修改样式时,都是对 HTML 元素对象的 class 属性进行修改,可以使用 Chrome 中的“开发者工具”进行检查,并将其作为界面调试的重要手段。

5.3.3 style 对象

使用 className 和 classList 对象已经可以很方便地修改 HTML 元素对象的样式,但这种方式只适合于样式的值是固定不变的,如果样式的值需要引入变量形式,则必须借助于每个 HTML 元素对象的 style 对象,用它来访问 HTML 元素对象的样式信息。

style 对象包含了与每个 CSS 样式对应的属性,只是格式略有不同:

- (1) 对于单个单词的 CSS 样式,以相同的名字属性来表示(例如,color 样式通过 style.color 表示);
- (2) 对于两个单词的 CSS 样式,则是通过去除横杠,将第一个单词加上首字母大写的第二个单词(例如: background-color 样式对应 style.backgroundColor)。表 5-5 列出了一些常用的 CSS 样式以及它们对应的 JavaScript 中 style 对象的属性表示。

表 5-5 CSS 样式与 style 属性的对应示例

CSS 样式	style 中的属性
background-color	style.backgorundColor
color	style.color
font	style.font
font-family	style.fontFamily
font-weight	style.fontWeight

使用 style 对象可以方便地获取 HTML 元素对象的 style 属性所定义的 CSS 样式值,但它无法获取在外部定义的 CSS 样式。

【例 5-16】 style 对象修改样式使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title> style 对象修改样式使用示例</title>
    <style>
      div{
        width: 150px;
        height: 150px;
        color: #fff;
        font - size: 20px;
        font - weight: bold;
        text - align: center;
      }
    </style>
  </head>
  <body>
    <div id = "mydiv" style = "background - color: red;">
      Hello HTML5
    </div>
    <span id = "desc1"></span><br />
    <span id = "desc2"></span>
    <script>
      var oDiv = document.getElementById("mydiv");
      var oSpan1 = document.getElementById("desc1");
      var oSpan2 = document.getElementById("desc2");
      oSpan1.innerHTML = "div 的背景色为"
        + oDiv.style.backgroundColor;
      oSpan2.innerHTML = "div 中的字体颜色为:"
        + oDiv.style.color;      //color 是在外部定义的,无法读取
      var w = 300;
      setTimeout(function(){
        oDiv.style.width = w + "px";
        oDiv.style.height = w + "px";
        oDiv.style.backgroundColor = "yellow";
        oDiv.style.color = "#000";
        oDiv.style.fontSize = "40px";
        oSpan1.innerHTML = "div 的背景色为"
          + oDiv.style.backgroundColor;
      },5000);
    </script>
  </body>
</html>
```

程序运行后,页面上出现一个红色的 div,里面含有白色的文字,5s 后使用 style 对象修改了 div 的大小和背景色,内部文字的大小和颜色也作了修改,如图 5-10 所示。

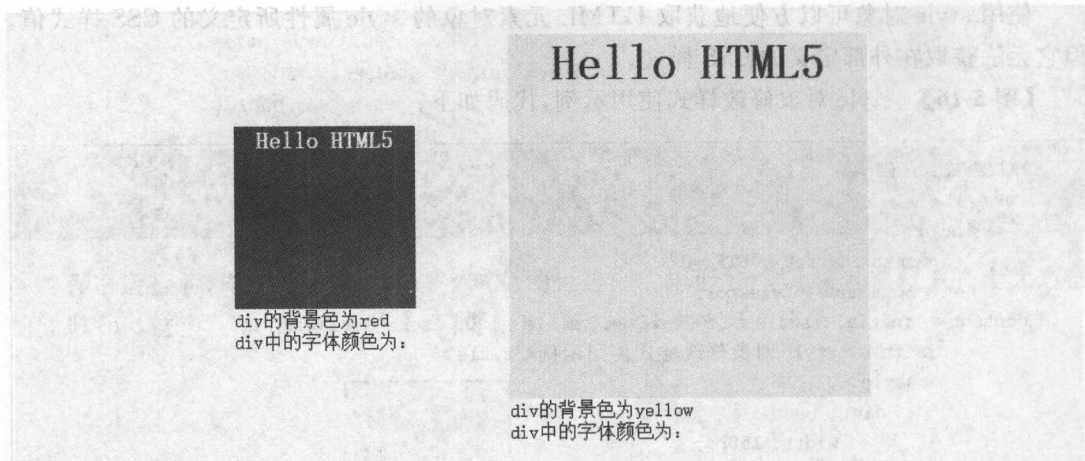


图 5-10 style 对象读取和设置样式



使用 style 对象设置样式时，实际就是修改 HTML 元素对象的 style 属性。

5.4 事件

JavaScript 是基于对象(object-based)的语言，基于对象的基本特征，就是采用事件驱动(event drive)。事件是在浏览器中可以被 JavaScript 检测到的行为，例如用户单击了按钮、移动了手指，都会触发相应的事件。用来响应某个事件的函数则称为事件处理程序，或者称为事件监听函数。

5.4.1 常用的一些事件

在页面浏览过程中，由鼠标或键盘、触摸屏会驱动一系列事件的激发，表 5-6 列出了一些常用的事件。

表 5-6 常用事件举例

归类	事件名称	描述
键盘	onkeydown	某个键盘的键被按下触发
	onkeypress	某个键盘的键被按住触发
	onkeyup	某个键盘的键被松开触发
鼠标	onmousedown	鼠标键按下触发
	onmousemove	鼠标被移动触发
	onmouseout	鼠标从某个 HTML 元素移开触发
	onmouseover	鼠标移动到某个 HTML 元素上触发
	onmouseup	松开鼠标键触发
	onclick	鼠标单击某个 HTML 元素触发
	ondblclick	鼠标双击某个 HTML 元素触发

发,它的语法格式为: 若事件监听函数需要返回值,则续表

归类	事件名称	描述
触摸	ontouchstart	当手指触摸屏幕时候触发,即使已经有一个手指放在屏幕上也会触发
	ontouchmove	当手指在屏幕上滑动的时候连续地触发
	ontouchend	当手指从屏幕上离开的时候触发
	ontouchcancel	当系统停止跟踪触摸的时候触发(例如有电话或短信时)
其他	onabort	图像加载被中断被触发
	onblur	HTML 元素失去焦点时触发
	onchange	内容发生改变时触发
	onerror	当加载文档或图像时发生错误时触发
	onfocus	HTML 元素获得焦点时触发
	onload	加载页面或图像时触发
	onresize	窗口调整尺寸时触发
	onunload	退出页面时触发
	onunload	退出页面时触发

5.4.2 内联属性监听事件

这种方法是指在 HTML 元素里面直接填写事件有关属性,属性值为相关 JavaScript 代码,即可在触发该事件的时候,执行属性值里面的代码。

【例 5-17】 使用 HTML 元素内联属性监听事件使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title>使用 HTML 元素内联属性监听事件</title>
  </head>
  <body>
    <input type = "button" value = "运行 JS 语句"
onclick = "alert('hello');"/>
    <input type = "button" value = "运行 JS 函数"
onclick = "eventTest();"/>
    <script>
      function eventTest(){
        alert("函数运行");
      }
    </script>
  </body>
</html>
```

在这个例子中,当单击按钮时,就会触发 click 事件,执行 onclick 属性中的 JavaScript 语句。显而易见,使用这种方法时,JavaScript 代码与 HTML 代码耦合在了一起,不便于维

护和开发,所以要尽量避免使用这种方法。

5.4.3 DOM 属性监听事件

使用 DOM 属性绑定事件可以解决代码的耦合问题,使用也比较简单,只需要在相应的节点对象上直接使用表 5-6 中的事件名称作为属性,赋予匿名函数或函数名即可。它比较简单易懂,而且有更好的兼容性。但是也有缺陷:因为直接赋值给对应事件属性,如果在后面代码中再次为节点对象绑定一个回调函数,会覆盖之前回调函数的内容。

【例 5-18】 使用 DOM 属性监听事件使用示例,代码如下:

```
<!DOCTYPE html >
<html >
  <head>
    <meta charset = "UTF-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>使用 DOM 属性监听事件</title>
  </head>
  <body>
    <input type = "button" value = "使用匿名函数" id = "mybutton1"/>
    <input type = "button" value = "使用函数名" id = "mybutton2"/>
    <script>
      var oInput1 = document.getElementById("mybutton1");
      var oInput2 = document.getElementById("mybutton2");
      oInput1.onclick = function(){
        alert("hello world");
      }
      //再次使用 DOM 属性监听,覆盖以前的监听事件
      oInput1.onclick = function(){
        alert("hello html5");
      }
      oInput2.onclick = sayHello;
      function sayHello(){
        alert("hello");
      }
    </script>
  </body>
</html>
```



使用 DOM 属性监听事件时,最容易犯错的地方是赋予的值不是函数名,而是执行函数,造成无法触发相应的事件,例如:

```
oInput2.onclick = sayHello(); //错误,sayHello 返回值是 undefined
```

5.4.4 标准的事件监听函数

在 HTML5 App 开发中,一般都是使用标准的事件监听函数来实现对某个事件的触

发,它的语法格式为:

```
addEventListener(eventName, callback, usecapture);
```

事件监听函数同样也是对于 HTML 元素对象使用,当监听到有相应事件发生的时候,调用 callback 回调函数。至于 usecapture 这个参数,表示该事件监听是在“捕获”阶段中监听(设置为 true)还是在“冒泡”阶段中监听(设置为 false)。usecapture 一般设置为 false。

标准的事件监听函数和前面的内联方式以及 DOM 方式都不同,如果对同一个 HTML 元素多次添加回调函数,这些回调函数会按先后次序依次执行。

如果想解除监听函数绑定,则需要使用 removeEventListener 方法:

```
removeEventListener (eventName, callback, usecapture);
```

需要注意的是,如果要移除监听函数绑定,绑定事件时的回调函数不能是匿名函数,必须是一个声明的函数,因为解除事件绑定时需要传递这个回调函数的引用(即函数名),才可以解开绑定。

【例 5-19】 使用标准的事件监听函数使用示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title>标准的事件监听函数</title>
    <style>
      div{
        width: 200px;
        height: 200px;
        background - color: yellow;
      }
      .newBk{
        background - color: red;
      }
    </style>
  </head>
  <body>
    <div id = "mydiv">
    </div>
    <span id = "desc"></span><br /><br />
    <input type = "button" value = "实现绑定" id = "btnAdd"/>
    <input type = "button" value = "解除绑定" id = "btnRemove"/>
    <script>
      var oInput1 = document.getElementById("btnAdd");
```



```

var oInput2 = document.getElementById("btnRemove");
var oDiv = document.getElementById("mydiv");
var oSpan = document.getElementById("desc");
//对 btnAdd 添加 click 事件监听 1
oInput1.addEventListener("click",function(){
    //对 div 添加 mouseover 和 mouseout 事件监听
    oDiv.addEventListener("mouseover",changeBkColor,false);
    oDiv.addEventListener("mouseout",changeBkColor,false);
},false);
//对 btnAdd 添加 click 事件监听 2
oInput1.addEventListener("click",function(){
    desc.innerText = "绑定成功,请将鼠标放在 div 上或移出 div";
},false);
//对 btnRemove 添加 click 事件监听 1
oInput2.addEventListener("click",function(){
    //对 div 移除 mouseover 和 mouseout 事件监听
    oDiv.removeEventListener("mouseover",changeBkColor,false);
    oDiv.removeEventListener("mouseout",changeBkColor,false);
},false);
//对 btnRemove 添加 click 事件监听 2
oInput2.addEventListener("click",function(){
    desc.innerText = "解除绑定";
},false);
//对 div 的样式类 newBk 实现 toggle
function changeBkColor()
{
    oDiv.classList.toggle("newBk");
}
</script>
</body>
</html>

```

在这个例子中,单击“实现绑定”按钮后,鼠标放在黄色的 div 上,div 会自动变成红色,鼠标移出 div,会恢复为黄色;当单击“解除绑定”按钮后,鼠标移动或移出 div 不再有效果出现,如图 5-11 所示。

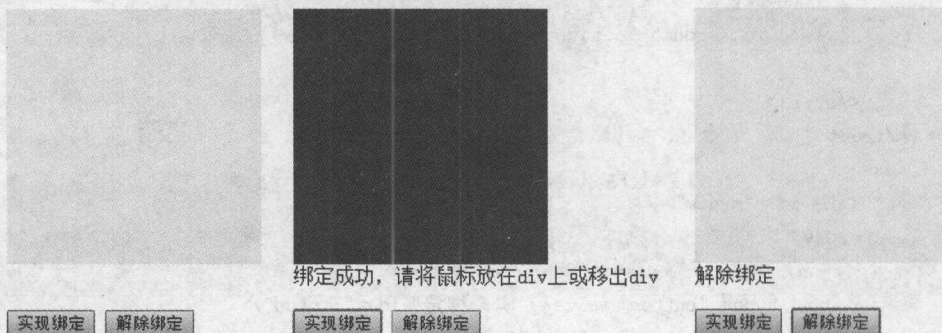


图 5-11 标准的事件监听函数

5.4.5 事件触发过程

前文大体介绍了事件是什么、如何监听并执行某些操作,但还未涉及事件触发的整个过程,下面来讨论事件的触发过程,它分为几个阶段:捕获阶段、目标阶段、冒泡阶段,如图 5-12 所示。

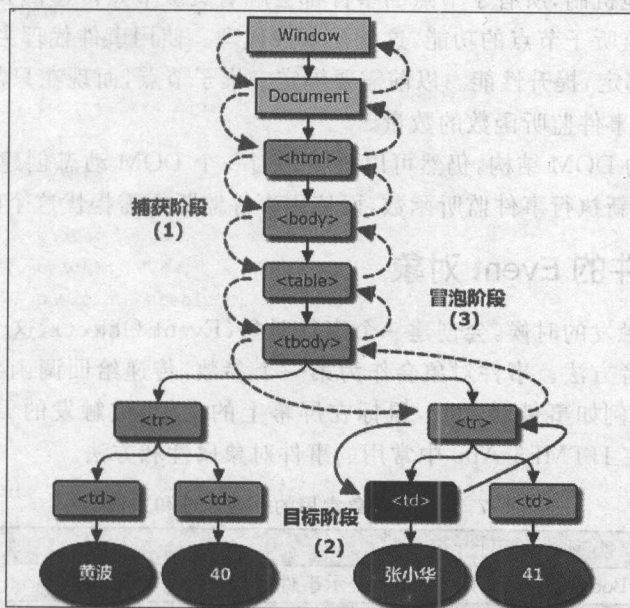


图 5-12 事件触发过程

1. 捕获阶段

当 DOM 树的某个节点发生了一些操作(例如单击、鼠标移动上去),就会有一个事件被触发。这个事件从 Window 发出,不断经过下级节点直到目标节点。在到达目标节点之前的过程,就是捕获阶段(Capture Phase)。

所有经过的节点,都会触发这个事件。捕获阶段的任务就是建立这个事件传递路线,以便后面冒泡阶段顺着这条路线返回 Window。

监听某个在捕获阶段触发的事件,需要在将标准的事件监听函数的参数 `usecapture` 设置为 `true`。

2. 目标阶段

当事件流到达事件触发目标节点那里,最终在目标节点上触发这个事件,这就是目标阶段(Target Phase)。需要注意的是,事件触发的目标总是最底层的节点。例如单击表格中的一段文字,以为事件目标节点在 `<td>` 上,但实际上触发在它的文字子节点上。

3. 冒泡阶段

当事件达到目标节点之后,就会沿着原路返回,由于这个过程类似水泡从底部浮到顶部,所以称作冒泡阶段(Bubbling Phase)。在实际使用中,并不需要把事件监听函数准确绑定到最底层的节点也可以正常工作。例如为 `<td>` 绑定单击时的回调函数时,无需为它下面的所有子节点全部绑定单击事件,只需要为 `<td>` 这一个节点绑定即可。因为发生它子节点

的单击事件,都会冒泡上去,发生在<td>上。

所以在使用标准的事件监听函数时,usecapture 参数一般设为 false,这样监听事件时只会监听冒泡阶段发生的事件。



IE 浏览器比较特殊, IE8 以前的版本不支持在捕获阶段监听事件。

因为事件有冒泡机制,所有子节点的事件都会顺着父级节点传递回去,所以可以通过监听父级节点来实现监听子节点的功能,这就是事件代理。使用事件代理主要有两个优势:

(1) 减少事件绑定,提升性能。以前需要绑定一堆子节点,而现在只需要绑定一个父节点即可,减少了绑定事件监听函数的数量。

(2) 动态变化的 DOM 结构,仍然可以监听。当一个 DOM 动态创建之后,不会带有任何事件监听,除非重新执行事件监听函数,而使用事件监听无需担忧这个问题。

5.4.6 事件的 Event 对象

当一个事件被触发的时候,会创建一个事件对象(Event Object),这个对象里面包含了一些有用的属性或者方法。事件对象会作为第一个参数,传递给回调函数。事件对象包含了很多有用的信息,例如事件触发时,鼠标在屏幕上的坐标、被触发的 DOM 详细信息等,表 5-7 列出了一些在 HTML5 App 中常用的事件对象属性和方法。

表 5-7 Event 对象常用的一些属性和方法

属性/方法	类型	可读/可写	描述
cancelable	Boolean	只读	表示事件能否取消
cancelBubble	Boolean	只读	表示事件冒泡是否取消
clientX	Number	只读	事件发生时,鼠标或触摸点在客户端区域(不包含工具栏、滚动条等)的 x 坐标
clientY	Number	只读	事件发生时,鼠标或触摸点在客户端区域(不包含工具栏、滚动条等)的 y 坐标
currentTarget	Object	只读	触发事件的 HTML 元素对象
eventPhase	Number	只读	事件所处阶段,0—捕获阶段,1—目标阶段,2—冒泡阶段
pageX	Number	只读	鼠标或触摸点相对于页面的 x 坐标
pageY	Number	只读	鼠标或触摸点相对于页面的 y 坐标
preventDefault()	Function	只读	阻止事件的默认为
screenX	Number	只读	相对于屏幕的鼠标 x 坐标
screenY	Number	只读	相对于屏幕的鼠标 y 坐标
stopPropagation()	Function	只读	阻止事件冒泡
target	Object	只读	引起事件的 HTML 元素对象
type	String	只读	触发的事件类型
isTrusted	Boolean	只读	事件是浏览器触发(用户真实操作触发),还是 JavaScript 代码触发的

【例 5-20】 触摸事件和 Event 对象使用示例,代码如下:

```
<!DOCTYPE html>
<html>
```

```

<head>
  <meta charset = "UTF - 8">
  <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
  <title> Event 对象使用</title>
  <style>
    # showDiv{
      height: 200px;
      width: 100 % ;
      font - size: 20px;
    }
    # objDiv{
      background - color: red;
      width: 100px;
      height: 100px;
      position: absolute;
      left:200px;
    }
  </style>
</head>
<body>
  <div id = "showDiv"></div>
  <div id = "objDiv"></div>
  <script>
    //开始触摸时的 x 坐标
    var sx;
    //开始触摸时的 y 坐标
    var sy;
    var oDiv = document.getElementById("showDiv");
    var obj2 = document.getElementById("objDiv");
    function touches(ev) {
      switch(ev.type) {
        case 'touchstart':
          ev.preventDefault(); //阻止出现滚动条
          oDiv.innerHTML = 'Touch start(' +
ev.changedTouches[0].clientX + ', ' + ev.changedTouches[0].clientY + ')';
          sx = ev.touches[0].clientX - obj2.offsetLeft;
          sy = ev.touches[0].clientY - obj2.offsetTop;
          break;
        case 'touchmove':
          oDiv.innerHTML = 'Touch move(' +
ev.changedTouches[0].clientX + ', ' + ev.changedTouches[0].clientY + ')';
          ev.preventDefault(); //阻止出现滚动条
          var mx = ev.changedTouches[0].clientX - sx;
          var my = ev.changedTouches[0].clientY - sy;
          obj2.style.marginLeft = mx - 200 + "px";
          obj2.style.marginTop = my - 200 + "px";
          break;
        case 'touchend':

```



```
        oDiv.innerHTML = 'Touch end(' +  
        ev.changedTouches[0].clientX + ', ' + ev.changedTouches[0].clientY + ')';  
        break;  
    }  
}  
window.addEventListener("load", function() {  
    document.addEventListener('touchstart', touches, false);  
    obj2.addEventListener('touchmove', touches, false);  
    document.addEventListener('touchend', touches, false);  
}, false);  
</script>  
</body>  
</html>
```

程序运行后,在 Chrome 中打开“开发者工具”,把它切换成“移动设备模式”,鼠标自动模拟手指触点,按下鼠标左键,单击页面,页面显示触摸的 touchstart 事件被触发,按住鼠标左键对红色的 div 实现拖放操作,触发 touchmove 事件,当松开鼠标左键后,自动触发 touchend 事件,效果如图 5-13 所示。

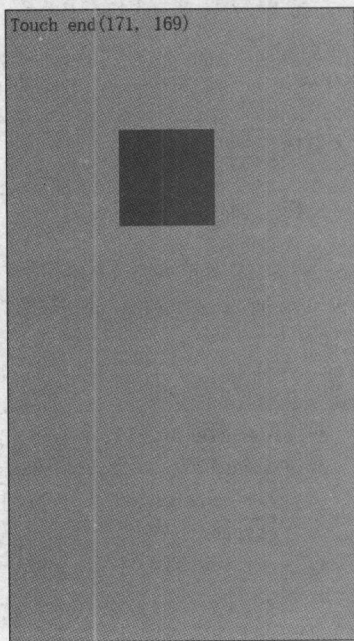


图 5-13 触摸事件和事件 Event 对象

小结

本章主要讲解了 JavaScript DOM 和事件的编程基础,介绍了 DOM 概念、document 对象的使用,详细讲解了 DOM 查找节点的各种方法,使用 DOM 进行属性操作、修改节点内部 HTML 和文字内容,创建节点、添加、插入、删除、替换、复制节点的各方法,介绍了事件

和一些常用事件以及如何实现事件监听。本章所讲解的内容主要应用在 HTML5 App 交互编程中,需要重点掌握。

习题

一、选择题

- document 对象中查找节点最有效的方法是()。
A. getElementsByTagName B. getElementById
C. getElementsByClassName D. querySelectorAll
- 要动态改变 div 节点对象中的内容,可以使用的方法有()。
A. innerText B. innerHTML C. split D. join
- 对于手指在触摸屏上移动,应该监听 HTML 节点对象的()事件。
A. ontouchstart B. ontouchmove C. ontouchend D. ontouchcancel
- 当使用 DOM 进行样式编程时,如果样式的值需要使用变量,应该使用()对象。
A. className B. classList C. style D. css
- 当需要频繁添加子节点时,创建子节点应该采用()方法,再一次性附加到父节点。
A. createElement B. createTextNode
C. createNode D. createDocumentFragment

二、判断题

- document.querySelector 会返回所有匹配 CSS 选择器的节点对象集合。()
- 使用 style 对象不能读取 HTML 元素对象 style 属性中定义的样式。()
- 为提高效率,查找固定的 HTML 元素节点对象最好不要放在循环中。()
- 为 HTML 元素增加自定义 data-属性,浏览器会报错。()
- cloneNode 的深复制和浅复制没有区别。()

三、填空题

- DOM 的英文全称是_____,中文意思是_____。
- 使用 DOM 编程,可以对节点对象_____、_____、_____。
- 使用 classList 对象可以为 HTML 元素对象_____、_____、_____样式类。
- 事件的触发过程分为_____、_____、_____三个阶段。
- JavaScript 中,每个事件的处理函数都有一个_____对象作为参数,它代表事件的状态,例如发生事件中的元素、触点的位置等。

四、简答题

常用的事件有哪些? 如何实现事件监听?

五、编程题

- 样式编程练习

有如下代码的页面:

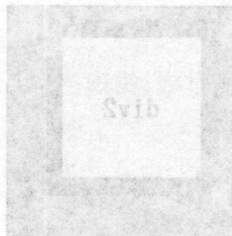


图 5-12 DOM 元素

图 5-14 事件

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <meta name = "viewport"
content = "initial - scale = 1.0,maximum - scale = 1.0, user - scalable = no" />
    <title></title>
    <style>
      div{
        margin: 0 auto;
        text-align: center;}
      #content div{
        width: 150px;
        height: 150px;
        margin-bottom: 20px;
        font-weight: bold;
        font-size: 30px;
        line-height: 150px;
      }
      body{
        background-color: #CCCCCC;
      }
    </style>
  </head>
  <body>
    <div id = "content">
      <div>div1</div>
      <div>div2</div>
      <div>div3</div>
    </div>
  </body>
</html>
```

用 DOM 实现样式编程,让 content 中的子 div 产生不同的背景颜色,并实现图 5-14 中的 div 的重叠效果。

2. DOM 编程练习

模拟实现上拉刷新和下拉加载效果,初始页面如图 5-15 所示。

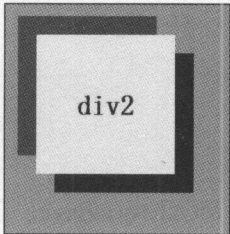


图 5-14 样式编程练习

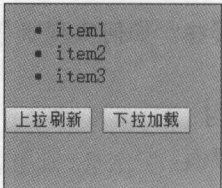


图 5-15 DOM 编程练习

页面代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>DOM 编程练习</title>
    <style>
      body{
        background-color: #ccc;
      }
    </style>
  </head>
  <body>
    <ul id="dataList">
      <li>item1</li>
      <li>item2</li>
      <li>item3</li>
    </ul>
    <input type="button" value="上拉刷新" id="btnPullUp"/>
    <input type="button" value="下拉加载" id="btnPullDown"/>
  </body>
</html>
```

用 DOM 编程及事件实现图 5-16 的效果，每单击一次按钮实现 3 个列表项的增加。

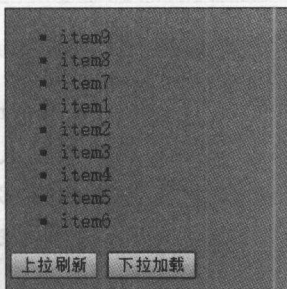


图 5-16 上拉刷新和下拉加载效果

第 6 章

CHAPTER 6

jQuery 编程基础

学习目标

- 掌握 jQuery 的各种自定义选择器。
- 掌握 jQuery 与 DOM 对象的转换。
- 掌握 jQuery 的页面载入事件和对各种事件的监听。
- 掌握使用 jQuery 的遍历方法。
- 掌握使用 jQuery 完成各种 DOM 交互。
- 掌握对 jQuery 的功能扩展。

jQuery 是业界目前最优秀的 JavaScript 框架之一,它拥有便捷的语法形式和丰富的插件,开发效率极高,在网站项目或 HTML5 App 中备受欢迎。本章将针对 jQuery 在实际开发中需要重点掌握的技术进行详细讲解,并结合前面的知识,使用 jQuery 完成一个简单的翻牌游戏。

6.1 jQuery 介绍

jQuery 是一个 JavaScript 脚本库(<http://www.jquery.com>),它的核心理念是 Write Less, Do More(写得更少,做得更多)。jQuery 是由美国人 John Resig 于 2006 年 1 月在纽约的 BarCamp 发布的,它吸引了来自世界各地的众多 JavaScript 高手加入,由 Dave Methvin 率领团队进行开发的。如今, jQuery 已经成为全球最流行的 JavaScript 库,在世界前 10000 个访问最多的网站中,有超过 55% 的网站在使用 jQuery。

jQuery 是基于 MIT 许可协议,是完全免费和开源的 JS 类库。jQuery 的语法设计可以使开发更加便捷,例如操作文档对象、选择 DOM 元素、制作动画效果、事件处理、使用 Ajax 以及其他功能。除此以外, jQuery 还提供各种 API 让开发者编写插件。其模块化的使用方式使开发者可以很轻松地开发出功能强大的静态或动态网页。在移动互联网时代,针对触屏的智能手机和平板电脑,美国的 Media Temple 公司联合多家移动设备厂商以及软件企业共同开发了基于 jQuery 的前端开发框架 jQuery Mobile(详见 <http://www.jquerymobile.com>),这个框架基于 HTML5 技术,可以支持响应式布局的网站和 App 开发。

jQuery 有几大特性:

(1) 轻量级,压缩过的 .js 脚本只有 32K,支持 AMD(Asynchronous Module Definition),即“异步模块定义规范”;

(2) 支持 CSS3 选择器;

(3) 兼容所有的浏览器,只要使用 jQuery 框架,基本不用担心代码在各浏览器上的兼容性问题;

(4) 有丰富的插件支持,毫不夸张地讲,基本上目前在网页上能看到的各种特效,只要在百度上搜索,都可以找到很多相应的插件支持,而且实现这些效果的插件代码量非常小。

正是因为这些特性,使得 jQuery 在各开发公司中成为前端程序员必备技能之一。

6.2 使用 jQuery

jQuery 脚本库实际上就是一个 .js 文件。在项目中使用 jQuery 库有两种方式,一种是下载脚本库,并放到本地项目中,这种方法主要用在 App 开发中;另一种是采用 CDN(内容分发网络)引用它,这种方法主要用于 Web 开发中。

jQuery 的下载网址为 <http://www.jquery.com/download/>,目前最新版本为 3.1.1,在项目开发中可以使用未压缩版本,以便于调试,文件名为 jquery-3.1.1.js,而在产品正式发布时采用压缩版本 jquery-3.1.1.min.js,以便减少 App 安装包尺寸和提高页面加载速度。如果是下载并加入项目,则在页面中使用 <script> 标签将 jQuery 脚本库引入页面,例如:

```
<script src="scripts/jquery-3.1.1.min.js"></script>
<script>
    //JavaScript 代码
    ...
</script>
```

另外,网络上目前有很多 jQuery 的 CDN 加速节点,甚至连 jQuery 官网也有提供,但是最好选择一个距离较近的节点使用,如果网站主要对国内用户,那最好就选择新浪的 CDN 加速节点,如果是面向国外用户,那自然是选择 Google 的 CDN 加速节点,要注意 CDN 中的 jQuery 不一定是最新版本,例如:

```
<script
src="http://lib.sinaApp.com/js/jquery/3.1.0/jquery-3.1.0.min.js">
</script>
//其他版本可以在这里找地址 http://lib.sinaApp.com/?path=/jquery
```

6.3 使用 \$() 函数

jQuery 最强大的特性之一就是它能够简化在 DOM 中查找 HTML 元素对象。jQuery 语法是为 HTML 元素的选取编制的,查找到 HTML 元素对象后,就可以对元素执行某些操作,它的基本语法结构为:


```
$ (selector).action()
```

其中,selector 是 jQuery 的选择器字符串,\$ 符号代表 jQuery 对象,action 是 jQuery 对象自带的各种方法,通过 jQuery 对象实际的操作目标元素会非常简单,可以轻松地为 jQuery 对象绑定事件、添加漂亮的效果。

为了创建 jQuery 对象,就要使用 \$() 函数。所有能在样式表中使用的选择器字符串都可以传递给这个函数,然后就可以对所匹配的 HTML 元素对象应用 jQuery 的各种方法。

6.4 jQuery 的自定义选择器

除了各种 CSS 选择器之外,jQuery 还添加了独有的完全不同的自定义选择器。这些自定义选择器进一步增强了已经十分强大的 CSS 选择器,提供了在页面上查找 HTML 元素对象的新手段。

为了创建 jQuery 对象,就要使用 \$() 函数。所有能在样式表中使用的选择器字符串都可以传递给这个函数,随后就可以对匹配的 HTML 元素对象应用 jQuery 的各种方法。

jQuery 的自定义选择器都可以让我们从页面中查找到一个或多个 HTML 元素对象。自定义选择器通常跟在一个 CSS 选择器后面,基于已选择出的 HTML 元素集来查找元素。自定义选择器的语法与 CSS 中的伪类选择器语法相同,即以冒号“:”开头。下面将介绍 jQuery 的各种自定义选择器。

本书的配套资源包提供了一个 jQuery 选择器的练习页面,如图 6-1 所示。在这个页面中,右边是示范的 HTML 页面,左边是各种选择器的测试,当鼠标单击某个选择器后面的 toggle 按钮时,匹配结果会以黄色在右边页面上醒目显示,若鼠标点击 documentation 按钮时,则会打开 jQuery 的官方 API 说明。

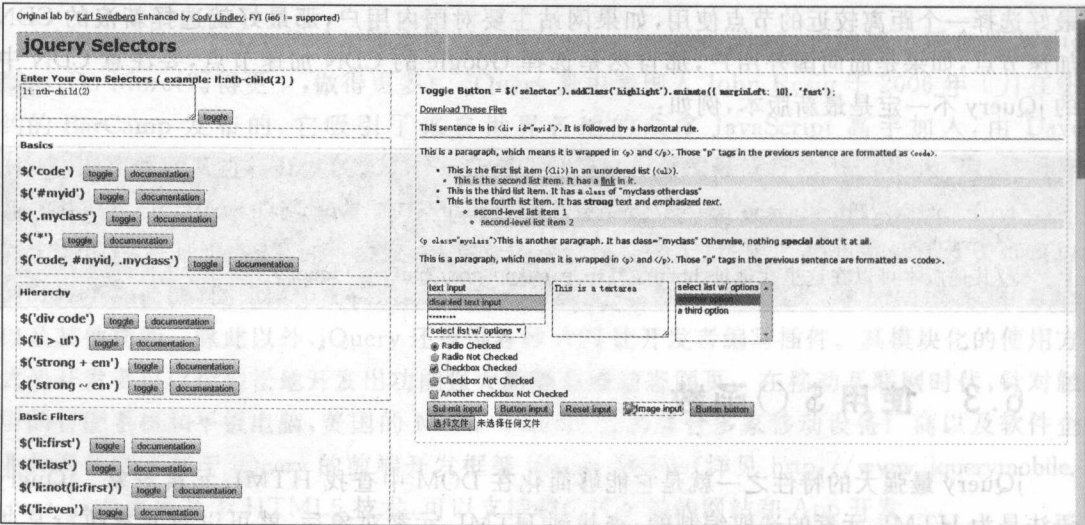


图 6-1 jQuery 选择器练习页面

6.4.1 基本过滤器

表 6-1 列举了 jQuery 的各种基本过滤器的使用。

表 6-1 jQuery 的基本过滤器

名称	说 明	举 例
:first	匹配找到的第一个元素	查找表格的第一行: \$("tr:first")
:last	匹配找到的最后一个元素	查找表格的最后一行: \$("tr:last")
:even	匹配所有索引值为偶数的元素,从 0 开始计数	查找表格的 1、3、5...行: \$("tr:even")
:odd	匹配所有索引值为奇数的元素,从 0 开始计数	查找表格的 2、4、6...行: \$("tr:odd")
:eq(index)	匹配给定索引值的元素,从 0 开始计数	查找表格的第 3 行: \$("tr:eq(2)")
:gt(index)	匹配所有大于给索引值的元素,从 0 开始计数	查找表格的第 3 行后面的行: \$("tr:gt(2)")
:lt(index)	匹配所有小于给索引值的元素,从 0 开始计数	查找表格的第 1 行和第 2 行: \$("tr:lt(2)")
:header	匹配所有 h1,h2,h3 一类的 header 标签	查找所有标题行: \$("tr:header")
:animated	匹配所有正在执行动画效果的元素	查找动画的 div 元素: \$("div:animated")

6.4.2 内容过滤器

表 6-2 列举了 jQuery 的各种内容过滤器的使用。

表 6-2 jQuery 的内容过滤器

名 称	说 明	举 例
:contains(text)	匹配包含给定文本的元素	查找所有包含"html5"的 div 元素: \$("div:contains('html5')")
:has(selector)	匹配含有选择器所匹配的元素	查找包含有 p 元素的 div 元素: \$("div:has(p)")
:parent	匹配至少有一个子节点的元素,包括文本节点	查找有子节点的 p 元素: \$("p:parent")

6.4.3 可见性过滤器

表 6-3 列举了 jQuery 的各种可见性过滤器的使用。

表 6-3 jQuery 的可见性过滤器


名称	说 明	举 例
:hidden	匹配所有不可见元素	查找所有隐藏的 div: \$("div:hidden")
:visible	匹配所有可见元素	查找所有可见的表格行: \$("tr:visible")

6.4.4 表单选择器

表 6-4 列举了 jQuery 的各种表单选择器的使用。

表 6-4 jQuery 的表单选择器

名称	说 明	举 例
:input	匹配所有的 input、textarea、button 元素	查找所有的 input 元素: \$("input")
:text	匹配所有的输入文本框	查找所有文本框: \$("input:text")
:password	匹配密码框	查找所有文本框: \$("input:password")
:radio	匹配单选框	查找所有单选框: \$("input:radio")
:checkbox	匹配复选框	查找所有复选框: \$("input:checkbox")
:submit	匹配提交按钮	查找所有提交按钮: \$("input:submit")
:image	匹配图片按钮	查找所有的图片按钮: \$("input:image")
:reset	匹配重置按钮	查找所有重置按钮: \$("input:reset")
:button	匹配普通输入按钮	查找所有普通按钮: \$("input:button")
:file	匹配上传按钮	查找所有上传文件按钮: \$("input:file")
:selected	匹配所有选中的 option 元素	查找所有选中的元素: \$("select option:selected")

 jQuery 会尽可能地使用浏览器原生的 DOM 选择器引擎去查找 HTML 元素。但如果使用 jQuery 的自定义选择器时,就无法使用速度最快的原生方法了。所以在实际开发中,为了确保程序性能,不要频繁地使用自定义选择符。


6.5 jQuery 对象与 DOM 对象的转换

所有的选择器和多数 jQuery 方法都能返回 jQuery 对象,这正是希望的结果,但是在有些情况下,我们希望在代码中能直接访问 DOM 对象。要把 jQuery 对象转换成标准的 DOM 对象, jQuery 提供了一个很简单的方式:在选择器后面直接使用方括号,这种方法与访问 DOM 元素数组很类似,使用方括号就像剥掉了 jQuery 的包装而直接露出节点列表,而方括号中的索引则相当于从中取出了原本的 DOM 对象,例如下面的代码:

```
var $div = $("#mydiv");    //取得 jQuery 对象
Var oDiv = $div[0];        //取得对应的标准的 DOM 对象
```

对于标准的 DOM 对象,由于只有有限的属性和方法,如果想应用 jQuery 已包装好的属性和方法,同样也可以将 DOM 对象转换成 jQuery 对象,转换的方式可以直接采用 \$() 方法,例如下面的代码:

```
var oDiv = document.getElementById("mydiv"); //取得 DOM 对象
Var $div = $(oDiv);                          //转换成相应的 jQuery 对象
```

 为了区分 DOM 对象和 jQuery 对象变量,一般在 jQuery 对象变量加上 \$,代表这是 jQuery 对象,可以直接使用 jQuery 定义好的属性和方法。

6.6 jQuery 对事件的处理

jQuery 它不仅提供了更优雅的事件处理语法,而且增强并扩展了基本的事件处理机制,在以下内容中会简单介绍它对事件的处理。

6.6.1 页面加载后执行

jQuery 提供了一个 `$(document).ready()` 方法来实现基于页面加载实现任务,这和原生的 `window.onload` 事件类似,但这个方法又优于 `window.onload`。例如,一个页面中如果包含多张图片,若通过 `window.onload` 事件来处理页面时,必须等这些图片全部下载完成,而使用 jQuery 注册的事件处理程序,则会在 HTML 页面下载完成并解析成 DOM 树之后,代码就可以运行了。另一方面,如果使用 `window.onload` 方式,如果多次监听事件,它只会保存最后一次指定的函数引用,而 jQuery 内部实现了一个事件队列,它每次调用这个方法时,都会向内部的队列中添加一个新函数,当页面加载完成后,所有函数都会按注册的顺序依次执行。

`$(document).ready()` 方法实际上是基于 `document` 这个 DOM 元素对象构建而成的 jQuery 对象,在它上面调用 `ready()` 方法。jQuery 中还提供了另外一种简写的方式,这两种方式是等效的,代码如下:

```
$(document).ready(function(){
    //执行代码
});
//等效的简写方式:
$(function(){
    //执行代码
});
```

【例 6-1】 `window.onload` 和 jQuery 的页面加载后执行使用示例,代码如下:

```
<!DOCTYPE html >
<html >
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title>window.onload 与 $(document).ready 方法使用</title>
  </head>
  <body>
    <script src = "scripts/jquery - 3.1.1.min.js" ></script>
    <script>
      function doSomething(){
        alert("something");
      }
    </script>
  </body>
</html>
```

```
function doOtherthing(){
    alert("otherthing");
}
//只能执行 doOtherthing
window.onload = doSomething;
window.onload = doOtherthing;
//先执行 doSomething,再执行 doOtherthing
$(document).ready(function(){
    doSomething();
});
$(function(){
    doOtherthing();
})
</script>
</body>
</html>
```



在使用 jQuery 编程时,一定要保证页面已引入了 jQuery 脚本库,否则页面会提示“\$ is not defined”错误。

6.6.2 jQuery 事件监听

表 6-5 列举了 jQuery 在事件监听中常用的一些方法。

表 6-5 jQuery 事件监听的常用方法

方 法	描 述
bind()	向匹配元素附加一个或更多事件处理函数
blur()	监听元素对象的 onblur 事件
change()	监听元素对象的 onchange 事件
click()	监听元素对象的 onclick 事件
dblclick()	监听元素对象的 ondblclick 事件
error()	当元素遇到错误(没有正确载入)时,监听 onerror 事件
focus()	监听元素对象的 onfocus 事件
hover()	一个模仿悬停事件(鼠标移动到一个对象上面及移出这个对象)的方法,当鼠标移动到元素上面时,会触发指定的第一个函数,移出这个元素时,会触发指定的第二个函数
keydown()	监听元素对象的 onkeydown 事件
keypress()	监听元素对象的 onkeypress 事件
keyup()	监听元素对象的 onkeyup 事件
mousedown()	监听元素对象的 onmousedown 事件
mouseenter()	只有在鼠标指针穿过元素时,才会触发 mouseenter 事件
mouseleave()	只有在鼠标指针离开元素时,才会触发 mouseleave 事件
mousemove()	监听元素对象的 onmousemove 事件
mouseout()	监听元素对象的 onmouseout 事件,不论鼠标指针离开元素或其子元素,都会触发 mouseout 事件

续表

方 法	描 述
mouseover()	监听元素对象的 onmouseover 事件,不论鼠标指针穿过元素或其子元素,都会触发 mouseover 事件
mouseup()	监听元素对象的 onmouseup 事件
on()	给元素绑定一个或多个事件的事件处理函数
off()	给元素移除一个或多个用 on 绑定的事件处理函数
one()	向元素添加只能执行一次的事件处理函数
resize()	主要用于监听 window 对象的 onresize 事件
scroll()	监听元素的 onscroll 事件
select()	监听元素的 onselect 事件
submit()	监听 form 表单的 onsubmit 事件
trigger()	触发元素的某个事件
unbind()	移除由 bind()方法添加的一个或多个事件处理函数

从表 6-5 中可以看出,大多数事件在 jQuery 中都能找到一个等效的方法,不同的是,这里使用的是方法,方法的参数是函数入口点(匿名函数或已定义函数的函数名),在 jQuery 中,它的事件监听方式是采用自带的事件队列机制。

【例 6-2】 jQuery 对按钮单击事件进行监听使用示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>jQuery 对按钮单击事件进行监听</title>
  </head>
  <body>
    <input type="button" value="按钮事件的监听" id="mybutton"/>
    <input type="button" value="只执行一次单击的按钮"
id="mybutton1"/>
    <script src="scripts/jquery-3.1.1.min.js"></script>
    <script>
      $(function(){
        //按钮的单击事件,使用 click 方法,并传递数据
        $("#mybutton").click("数据 1",function(e){
          console.log("按钮单击事件 1,参数的值:"+e.data);
        });
        //按钮的单击事件,使用 click 方法,不传递数据
        $("#mybutton").click(function(){
          console.log("按钮单击事件 2");
        });
        //按钮的单击事件,使用 on 方法,并传递数据
        $("#mybutton").on("click", {"myvalue": "数据 2"},
function(e){

```



```

        console.log("按钮单击事件 3, 参数的值:"
            + e.data.myvalue);
    });
    //按钮的单击事件, 使用 on 方法, 不传递数据
    $("#mybutton").on("click", function(){
        console.log("按钮单击事件 4");
    });
    //按钮的单击事件, 使用 bind 方法, 并传递数据
    $("#mybutton").bind("click", "数据 3", function(e){
        console.log("按钮单击事件 5, 参数的值:" + e.data);
    });
    //按钮的单击事件, 使用 bind 方法, 不传递数据
    $("#mybutton").bind("click", function(){
        console.log("按钮单击事件 6");
        $(this).unbind("click");
    });
    //按钮的单击事件只执行一次
    $("#mybutton1").one("click", function(){
        alert("只能执行一次单击");
    });
}
</script>
</body>
</html>

```

在这个例子中, 以按钮的单击事件监听为例, 示范了如何使用 jQuery 进行相应的事件监听, 使用了 3 种方式:

- (1) 直接使用 click 方法;
- (2) 使用 on 方法对 click 事件进行监听;
- (3) 使用 bind 方法对 click 事件进行监听。

从本质上来讲, 这三种方式没有太大的区别, 在进行事件监听时都可以传递参数, 并在事件中通过事件参数对象的 data 属性进行读取。事件的监听也会自动借助事件队列进行叠加。

【例 6-3】 jQuery 事件监听其他用法示例, 代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
    <title> jQuery 事件监听其他用法</title>
    <style>
      #mydiv{
        background-color: red;
        width: 150px;

```

```

        height: 150px;
    }
</style>
</head>
<body>
    <div id="mydiv"></div>
    <script src="scripts/jquery-3.1.1.min.js"></script>
    <script>
        $(function(){
            $("#mydiv").bind({
                "click":function(){
                    console.log("div 被点击了");
                },
                "mouseover":function(){
                    console.log("鼠标进入了 div");
                },
                "mouseout":function(){
                    console.log("鼠标离开了 div");
                }
            });
            $("#mydiv").trigger("click");
        })
    </script>
</body>
</html>

```

这个例子运行如图 6-2 所示,页面一载入,在控制台就会输出“div 被点击了”,这是使用 trigger 方法的结果,它可以用来自动触发元素的指定事件。在这里,我们也看到了对于元素可以使用 bind 方法一次性实现多个事件的监听,它的参数是一个 JSON 对象;当鼠标进入和移出红色 div 时,控制台会自动进行相应的输出。

jQuery 中还提供了一个类似于 CSS 中: hover 选择器的 hover 方法,这个方法能模拟鼠标的悬停操作,它需要两个函数作为参数,分别用来监听元素的 onmouseover 和 onmouseout 事件,例 6-3 修改后的代码如下:

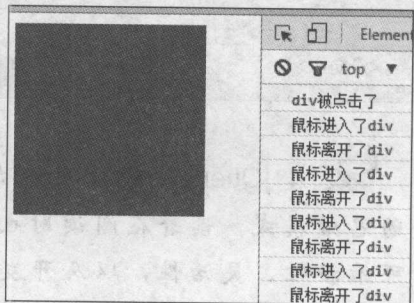


图 6-2 jQuery 事件监听的其他用法

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport"
            content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
        <title> jQuery 事件监听其他用法</title>
    </head>
    <body>
        <div id="mydiv"></div>
        <script src="scripts/jquery-3.1.1.min.js"></script>
        <script>
            $(function(){
                $("#mydiv").bind({
                    "click":function(){
                        console.log("div 被点击了");
                    },
                    "mouseover":function(){
                        console.log("鼠标进入了 div");
                    },
                    "mouseout":function(){
                        console.log("鼠标离开了 div");
                    }
                });
                $("#mydiv").trigger("click");
            })
        </script>
    </body>
</html>

```



```

<style>
    #mydiv{
        background-color: red;
        width: 150px;
        height: 150px;
    }
</style>
</head>
<body>
    <div id="mydiv"></div>
    <script src="scripts/jquery-3.1.1.min.js"></script>
    <script>
        $(function(){
            $("#mydiv").bind({
                "click":function(){
                    console.log("div 被点击了");
                }
            });
            $("#mydiv").trigger("click");
            $("#mydiv").hover(function(){
                console.log("鼠标进入了 div");
            },function(){
                console.log("鼠标离开了 div");
            })
        })
    </script>
</body>
</html>

```



在 jQuery 中，使用链式写法编写代码较为流行，它让代码更贴近作者的思维模式，读者在阅读时也更容易理解代码的含义，可以有效地提高系统的可维护性、灵活性，以及开发效率。

只要 HTML 元素对象不变，就可以一直调用 jQuery 的方法，它与 jQuery 的核心理念“Write less, Do more(写得更少，做得更多)”不谋而合。对例 6-3 的代码再次修改，精简成一句话的链式语句：

```

$("#mydiv").click(function(){
    console.log("div 被点击了");
}).hover(function(){
    console.log("鼠标进入了 div");
},function(){
    console.log("鼠标离开了 div");
}).trigger("click");

```


6.7 jQuery 遍历方法

jQuery 中提供了一个 each 方法,使用它可以遍历对象、数组,并进行处理,而不需要使用繁琐的循环语句。

6.7.1 遍历 HTML 元素对象

当使用 \$() 方法查找到匹配的 HTML 元素后,可以使用 each 方法进行遍历和操作相应的对象,它的语法形式如下:

```
$(selector).each(function([index]){
    //处理代码,this 关键字指向当前的对象,index 代表当前序号,可选参数
});
```

【例 6-4】 jQuery 的 each 方法对 HTML 元素对象的遍历,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0,maximum - scale = 1.0, user - scalable = no" />
    <title>each 方法遍历 HTML 元素对象</title>
    <style>
      img{
        width: 180px;
        height: 90px;
      }
    </style>
  </head>
  <body>
    <div id = "mydiv">
      <img src = "../img/163.png"><br>
      <img src = "../img/baidu.png"><br>
      <img src = "../img/sina.png" />
    </div>
    <script src = "scripts/jquery - 3.1.1.min.js" ></script>
    <script>
      $(function(){
        $("img").each(function(index){
          alert(this.src);
          if(index!= 1){
            this.src = "../img/sina.png";
          }
        });
      });
    </script>
```

```

    </script>
  </body>
</html>

```

运行后,页面依次弹出 3 个 img 标签的 src 属性,之后第一个和第三个图片都会变成网易的 Logo,结果如图 6-3 所示。在 each 方法的回调函数中,可以使用 this 关键字访问到当前遍历到的相应的 HTML 元素对象,它的参数 index 是可选的,代表当前遍历的元素的序号。



图 6-3 each 对 HTML 元素遍历前后效果

6.7.2 遍历数组对象

当 each 方法用于遍历数组时,它的语法形式如下:

```

$.each(array,function([index]){
    //处理代码,this 关键字指向当前的数组元素,index 代表当前序号,可选参数
});

```

【例 6-5】 jQuery 的 each 方法对数组对象的遍历,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>each 方法遍历数组</title>
  </head>
  <body>
    <script src="scripts/jquery-3.1.1.min.js"></script>
    <script>
      var myArray=["abc","def","ghi"];
      $.each(myArray,function(index){
        alert("第"+index+"项的值为:"+this);
      });
    </script>
  </body>
</html>

```

```
    });  
  </script>  
</body>  
</html>
```

6.7.3 遍历 JSON 对象属性

当 each 方法用于遍历 JSON 对象的属性时,它的语法形式如下:

```
$ .each(object,function(key){  
    //处理代码,key 代表属性  
});
```

【例 6-6】 jQuery 的 each 方法对 JSON 对象的遍历,代码如下:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset = "utf - 8">  
    <meta name = "viewport">  
    content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />  
    <title> each 方法遍历 JSON 对象属性</title>  
  </head>  
  <body>  
    <script src = "scripts/jquery - 3.1.1.min.js" ></script>  
    <script>  
      var data = {"name": "黄波", "age": 40, "course": "HTML5 应用开发"};  
      $ .each(data, function(key) {  
        alert(key + " = " + data[key]);  
      });  
    </script>  
  </body>  
</html>
```

6.8 jQuery DOM 交互

jQuery 对 JavaScript 中的标准 DOM 操作进行了封装,表 6-6 进行了归纳,列举了 jQuery 中与标准 JS 的 DOM 交互操作对应关系。

表 6-6 jQuery 中与标准 DOM 交互的对应

描述	标准 DOM 方法或属性	jQuery 中的方法
属性处理	getAttribute()、setAttribute()、removeAttribute()	attr()、removeAttr()
内容设置	innerHTML、innerText	html()、text()

续表

描述	标准 DOM 方法或属性	jQuery 中的方法
表单元素的值	value	val()
创建元素	createElement()	\$(“元素名”), 例如: \$(“<div></div>”)
附加元素	appendChild()	Append() 或 AppendTo() 实现尾部附加 prepend() 或 prependTo() 实现头部附加
插入元素	insertBefore(), insertAfter()	before(), after()
删除元素	removeChild()	remove(), empty()
替换元素	replaceChild()	replaceWith()
复制元素	cloneNode()	clone()
样式修改	className, classList, Add(), classList. Remove(), style, 样式	addClass(), removeClass(), toggleClass(), css()

6.8.1 操作 HTML 属性

在实际开发过程中,需要操作 HTML 元素的一些属性。jQuery 中使用 attr() 方法读取或设置 HTML 元素对象的任意属性,包括标准 DOM 属性和自定义属性,使用 removeAttr() 方法移除指定的属性。

语法形式如下:

```
.attr(name);           //读取某个属性的值  
.attr(name, value);    //设置某个属性的值  
.removeAttr(name);     //移除某个属性
```

【例 6-7】 jQuery 对 HTML 元素对象属性的处理,代码如下:

```
<!DOCTYPE html >  
<html >  
  <head >  
    <meta charset = "utf - 8">  
    <meta name = "viewport"  
    content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />  
    <title> jQuery 处理属性</title>  
  </head >  
  <body >  
    <input type = "text" value = "hello html5" id = "myinput"  
      data - myattr = "huangbo"/>  
    <script src = "scripts/jquery - 3.1.1.min. js" ></script>  
    <script >  
      $(function(){  
        //读取 value 属性  
        alert( $("#myinput").attr("value"));  
        //设置 type 属性的值为 button  
        $( "#myinput").attr("type", "button");  
      });  
    </script>  
  </body >  
</html >
```

```

        //移除 data-myattr 属性
        $("#myinput").removeAttr("data-myattr");
    });
</script>
</body>
</html>

```

程序运行后,会自动弹出输入框的 value 属性值“hello html5”,随后将输入框变成一个按钮。如果用 chrom 的“开发者工具”查看变化后的 HTML 源,会发现 data-myattr 属性已经移除。

在第 5 章中介绍过 data-自定义属性,可以使用它来方便地存储值而不会影响到界面显示,但它的缺点也很明显,用户也可以通过 Chrome 的“开发者工具”查看到该属性的值,从 jQuery 1.4 开始,jQuery 提供了一个 data 方法,用它来在 HTML 元素上存放值,即便使用“开发者工具”也无法查看到。这个方法兼容 data-属性,可以用来读取 HTML 元素的自定义属性值,但读取属性名字时,要去除 data-前缀,存储时则使用 jQuery 的内部数据缓存处理。

语法形式如下:

```

$(selector).data(name);
$(selector).data(name,value);

```

【例 6-8】 jQuery 的 data 方法存储和读取,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title> jQuery 的 data 方法存储和读取</title>
  </head>
  <body>
    <input type="text" value="hello html5" id="myinput"
data-myattr="huangbo"/>
    <script src="scripts/jquery-3.1.1.min.js"></script>
    <script>
      $(function(){
        //使用 data 方法来读取 data-myattr 属性
        alert($("#myinput").data("myattr"));
        //存储一个新的属性值,不可见
        $("#myinput").data("mynewattr","Study HTML5");
        //读取存储的属性值
        var x = $("#myinput").data("mynewattr");
        alert(x);
      });
    </script>
  </body>
</html>

```

```

    </script>
  </body>
</html>

```

6.8.2 操作表单元素的值

jQuery 中一个高效的方法是 `val()`，这是使用频率非常高的一个方法，使用这个方法可以轻松地对表单中各输入控件的值进行读取或设置。

语法形式如下：

```

.val(); //读取 HTML 元素对象的值
.val(值): //设置 HTML 元素对象的值

```

【例 6-9】 jQuery 的 `val` 方法读取和设置值，代码如下：

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title> jQuery 处理表单的值</title>
  </head>
  <body>
    <input type = "text" value = "huangbo" id = "myname" />
    <select id = "single">
      <option> Single1 </option>
      <option> Single2 </option>
    </select>
    <select id = "multiple" multiple = "multiple">
      <option selected = "selected"> Multiple1 </option>
      <option> Multiple2 </option>
      <option selected = "selected"> Multiple3 </option>
    </select><br/>
    <input type = "checkbox" name = "course" checked
      value = "JavaScript" /> JavaScript
    <input type = "checkbox" name = "course" value = "HTML5" /> HTML5 <br />
    <input type = "radio" name = "gender" value = "boy" /> 男
    <input type = "radio" name = "gender" value = "girl" checked /> 女
    <script src = "scripts/jquery-3.1.1.min.js"></script>
    <script>
      $(function() {
        alert( $("#myname").val()); //输出 "huangbo"
        alert( $("#single").val()); //输出 "Single1"
        //输出数组 Multiple1,Multiple3
        alert( $("#multiple").val());

```



```

        //输出"JavaScript"
        alert( $("input[name='course']:checked").val());
        //输出"girl"
        alert( $("input[name='gender']:checked").val());
        $("#myname").val("hello");           //修改输入框的值
        $("#single").val("Single2");         //修改单选框的值
        //修改多选框的值
        $("#multiple").val(["Multiple2", "Multiple3"]);

    });
</script>
</body>
</html>

```

运行后的效果如图 6-4 所示,可以看出,用 jQuery 的 val 方法可以轻松处理表单中各输入控件的值。

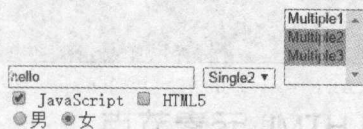


图 6-4 jQuery 处理表单输入控件的值

6.8.3 修改节点内容

在开发过程中,需要替换页面上的显示内容时, jQuery 提供了两个方法,一个是 html 方法,用来替换或读取页面上的 HTML 代码;另一个是 text 方法,用来替换或读取页面上的文字。它们的语法形式如下:

```

.html();           //读取内部 HTML 代码
.html(htmlContent): //设置内部 HTML 代码

.text();           //读取内部文字
.text(textContent): //设置内部文字内容

```

【例 6-10】 jQuery 的 html 和 text 方法,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title> jQuery 的 html 和 text 方法</title>
  </head>
  <body>
    <div id="content">

```

```

        <p>Hello World</p>
        <p>Hello HTML5</p>
    </div>
    <script src = "scripts/jquery-3.1.1.min.js"></script>
    <script>
        $(function(){
            //修改第 2 个 p 标签的文字
            $("#content p:eq(1)").text("HTML5 Study");
            //读取 content 的内部的所有文字
            alert($("#content").text());
            //读取 content 的内部所有 html 代码
            alert($("#content").html());
            //修改内部标签为图片显示
            $("#content").html("<img src = '../img/baidu.png'/>");
        });
    </script>
</body>
</html>

```

6.8.4 创建和添加 HTML 元素节点

在 jQuery 开发中,可以使用 jQuery 很轻松地创建新的 HTML 元素,但要注意的是,这只是在内存中创建了对象,并未添加到 DOM 树中,创建的语法形式为:

```
$ (htmlContent);
```

和传统的 JavaScript 相比,jQuery 封装了比较丰富的添加 HTML 元素节点的方法,这些方法的语法形式为:

. Append(content);	//在被选节点内部结尾部附加内容
. AppendTo(objectNode);	//将内容附加到被选元素内部尾部
. prepend(content);	//在被选节点内部结头部附加内容
. prependTo(objectNode);	//将内容附加到被选元素内部头部
. before(content);	//在被选元素节点前插入内容
. after(content);	//在被选元素节点后插入内容

【例 6-11】 jQuery 的创建和添加元素,代码如下:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8">
        <meta name = "viewport"
    content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
        <title> jQuery 创建和添加 HTML 元素</title>
        <style>

```



```

/* CSS 代码略, 请参考配套源代码 */
</style>
</head>
<body>
  <div id="mydiv">
  </div>
  <input type="button" value="附加到尾部" id="btnAppend"/>
  <input type="button" value="附加到头部" id="btnPrepend"/>
  <input type="button" value="在前面插入" id="btnBefore"/>
  <input type="button" value="在后面插入" id="btnAfter"/>
  <script src="scripts/jquery-3.1.1.min.js"></script>
  <script>
    $(function(){
      var $div = $("#mydiv");
      var count = 1;
      //附加到 div 尾部
      $('<input type="text" value="这是目标节点"
        id="myobj"/>').appendTo($div);
      //上下两句是等效的
      //$(div).append($('<input type="text" value="huangbo" id="myname"/>'));
      var $objNode = $("#myobj");
      $("#btnAppend").click(function(){
        var $input = createInput();
        $input.appendTo($div);
        //或写成 $div.append($input);
      });
      $("#btnPrepend").click(function(){
        var $input = createInput();
        $div.prepend($input);
        //或写成 $input.prependTo($div);
      });
      $("#btnBefore").click(function(){
        var $input = createInput();
        $objNode.before($input);
      });
      $("#btnAfter").click(function(){
        var $input = createInput();
        $objNode.after($input);
      });
      //创建一个新的 input 节点对象
      function createInput(){
        var $input = $('<input type="text"/>');
        $input.val(count++);
        return $input;
      }
    });
  </script>
</body>
</html>

```


页面运行后,可以分别单击页面上的“附加到尾部”“附加到头部”“在前面插入”“在后面插入”四个按钮分别查看效果,如图 6-5 所示。

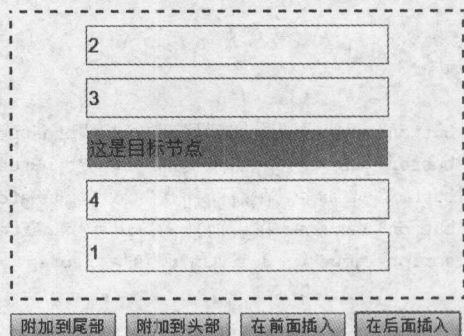


图 6-5 jQuery 创建和添加 HTML 元素

6.8.5 删除 HTML 元素节点

通过 jQuery 可以很容易地删除已有的 HTML 元素节点,它提供了两个方法以供使用,一个是 remove,另一个是 empty,它们的语法形式如下:

```
.remove();//删除目标节点及其子节点
.empty();//清空目标节点
```

【例 6-12】 jQuery 删除 HTML 元素,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title> jQuery 删除 HTML 元素</title>
    <style>
      /* CSS 代码略,请参看配套源代码 */
    </style>
  </head>
  <body>
    <ul id = "content">
      <li><span>这是 Item1</span>
        <input type = "button" value = "删除"/></li>
      <li><span>这是 Item2</span>
        <input type = "button" value = "删除"/></li>
      <li><span>这是 Item3</span>
        <input type = "button" value = "删除"/></li>
      <li><span>这是 Item4</span>
        <input type = "button" value = "删除"/></li>
```

```

<li><span>这是 Item5 </span>
    <input type="button" value="删除"/></li>
<li><span>这是 Item6 </span>
    <input type="button" value="删除"/></li>
</ul>
<input type="button" value="增加项目" id="btnAdd"/>
<input type="button" value="全部删除" id="btnRemoveAll"/>
<script src="scripts/jquery-3.1.1.min.js"></script>
<script>
    $(function(){
        var $ul = $("#content");
        //利用事件代理,防止新添加的 HTML 元素丢失事件
        $("#content").click(function(e){
            //移除当前项目
            $(e.target).parents("li").remove();
        });
        //增加新项目
        $("#btnAdd").click(function(){
            var count = $("#content li").length+1;
            var html = '<li><span>这是 Item' + count +
                '</span><input type="button" value="删除"/></li>';
            $ul.append(html);
        });
        //清空所有项目
        $("#btnRemoveAll").click(function(){
            $ul.empty();
        });
    });
</script>
</body>
</html>

```

页面运行后,效果如图 6-6 所示,可以单击每个项目中的“删除”按钮,从页面中删除这个按钮,也可以单击“全部删除”按钮,清空所有的项目。删除或清空后,也可以单击“增加项目”按钮添加项目。

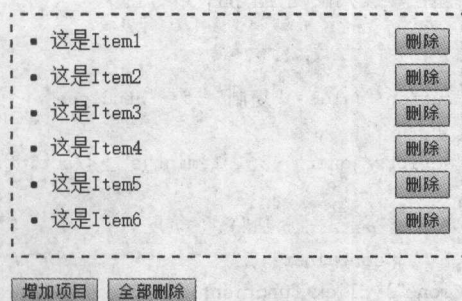


图 6-6 jQuery 删除 HTML 元素



这个例子中使用到了事件代理的概念(见第5章),如果只是监听按钮的 click 事件,对于新添加的项目会失效;例子中还用到了 jQuery 的 parents() 方法,代表所选节点对象的所有父节点,这个方法的参数可以再次使用选择器迅速定位到所需的父节点。

6.8.6 复制 HTML 元素节点

jQuery 中提供了一个 clone 方法,用于复制当前所匹配元素集合的一个副本,并以 jQuery 对象的形式返回,出于性能原因考虑,clone()方法不会复制某些表单元素的动态,例如,用户在< textarea>中输入的内容、用户在< select>中选择的选项。不过,< input>元素的动态将会被复制,例如用户在 text 中输入的内容、用户对 checkbox 的选中状态。它的语法形式为:

```
.clone(withDataAndEvents);
//withDataAndEvents 参数可以省略,默认为 false
```

【例 6-13】 jQuery 复制 HTML 元素,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title> jQuery 的复制 HTML 元素</title>
    <style></style>
    <style>
      /* CSS 代码略,请参看配套源代码 */
    </style>
  </head>
  <body>
    <ul id="container">
      <li></li>
    </ul>
    <div>
      <input type="button" value="复制" id="btnClone"/>
    </div>
    <script src="scripts/jquery-3.1.1.min.js"></script>
    <script>
      var $li= $("#container li");
      var $ul= $("#container");
      $("#btnClone").click(function(){
        $li.clone(true).appendTo($ul);
      });
      $("ul li").click(function(){
```



```
        alert( $(this).html() );
    });
</script>
</body>
</html>
```

运行效果如图 6-7 所示,可以用鼠标单击“复制”按钮,对项目不停地实现复制,并附加到页面上,由于 clone()方法使用了参数 true,li 的单项单击事件也被同时被复制了下来,如果使用 false 参数或不输入参数,复制生成的 li 单项不会调用单击事件。

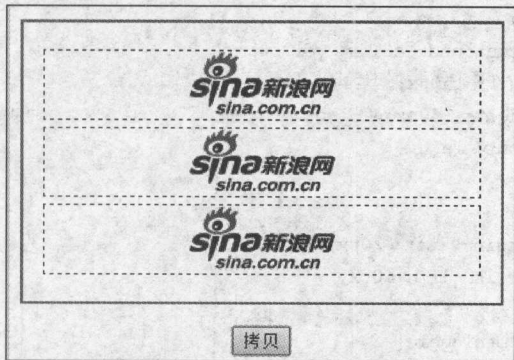


图 6-7 jQuery 复制 HTML 元素

6.8.7 修改样式

在实际的软件开发的过程中,如果要实现某种特效,利用 jQuery 控制元素的样式是最优的简单方法,HTML 元素样式的修改分为两种:一种是固定样式的修改,例如让元素实现旋转、隐藏等;另一种是非固定样式的修改,样式的值是变化的,例如需要根据手机的分辨率作样式的变化。

1. 固定样式的修改

对于固定样式的修改,需要在页面中事先制作 CSS 中的 class 选择器的样式,由 jQuery 提供的一些相应方法,对 HTML 元素的 class 属性进行操作,这些方法见表 6-7。

表 6-7 jQuery 固定样式修改方法

方 法	说 明	例 子
addClass	为每个匹配的 HTML 元素添加指定的 CSS 类名	<code>\$("# mydiv"). addClass("colorRed borderBlack")</code>
hasClass	判断 HTML 元素中是否应用了指定的 CSS 类名	<code>\$("# mydiv"). hasClass("borderBlack");</code>
removeClass	从所有匹配的元素中删除全部或者指定的 CSS 类名	<code>\$("# mydiv"). removeClass ();</code> <code>\$("# mydiv"). removeClass("colorRed");</code>
toggleClass	如果存在指定的 CSS 类名,则删除,不存在,则添加 CSS 类名	<code>\$("# mydiv"). toggleClass("colorRed borderBlack");</code>

【例 6-14】 jQuery 修改固定的样式,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title> jQuery 固定样式修改</title>
    <style></style>
    <style>
      #mydiv {
        background-color: yellow;
        width: 150px;
        height: 150px;
        float: left;
      }
      input{
        margin-left: 20px;
        margin-top: 40px;
      }
      input # btnToggle{
        width: 165px;
      }
      .borderBlack{
        border: 2px solid black;
      }
    </style>
  </head>
  <body>
    <div id = "content">
      <div id = "mydiv">
        </div>
        <input type = "button" value = "添加边框" id = "btnAdd"/>
        <input type = "button" value = "移除边框" id = "btnRemove"/>
        <input type = "button" value = "边框自动切换" id = "btnToggle"/>
      </div>
      <script src = "scripts/jquery-3.1.1.min.js"></script>
      <script>
        $(function(){
          var $div = $("#mydiv");
          $("#btnAdd").click(function(){
            if(! $div.hasClass("borderBlack")){
              $div.addClass("borderBlack");
            }
          });
          $("#btnRemove").click(function(){
            if( $div.hasClass("borderBlack")){
              $div.removeClass("borderBlack");
            }
          });
        });
      </script>
    </body>
  </html>
```

```

    }
  });
  $("#btnToggle").click(function(){
    $div.toggleClass("borderBlack");
  });
});
</script>
</body>
</html>

```

运行效果如图 6-8 所示,这个例子中,使用 CSS 的 class 选择器预先定义出边框。用鼠标单击“添加边框”按钮,对 div 实现 CSS 类的添加;单击“移除边框”按钮,对 div 实现 CSS 类的移除;单击“边框自动切换”按钮,自动进行 CSS 类样式的添加或移除。

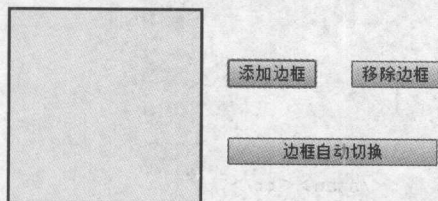


图 6-8 jQuery 固定样式修改

2. 非固定样式的修改

对于非固定样式的修改,由于无法事先制作相应的 CSS 样式,就要使用 jQuery 提供的 `css()` 方法,对 HTML 元素的 `style` 属性进行操作。从 jQuery 1.8 开始,当使用 CSS 属性在该方法中设置时,jQuery 将根据浏览器自动加上前缀(在适当的时候),例如 CSS 属性“`user-select`”,在 Chrome/Safari 浏览器中会设置为“`-webkit-user-select`”,Firefox 会使用“`-moz-user-select`”,IE10 将使用“`-ms-user-select`”。

CSS 的语法形式如下:

```

.css(name);           //读取相应 css 属性的值
.css(name,value);     //设置某个 css 属性
.css(properties);     //设置多个 CSS 属性的 JSON 对象

```

【例 6-15】 jQuery 修改非固定的样式,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0,maximum - scale = 1.0, user - scalable = no" />
    <title> jQuery 非固定样式修改</title>
    <style>

```


【例 6-14】 jQuery 修改固定的样式, 代码如下:

```
#mydiv {
    background-color: yellow;
    width: 50px;
    height: 50px;
    margin-bottom: 10px;
}
input[name = 'ratio'] {
    width: 30px;
    margin-bottom: 10px;
}
#btnLarge, #btnSmall {
    width: 60px;
    height: 30px;
    font-size: 20px;
}
</style>
</head>
<body>
    <div id = "mydiv">
    </div>
    <span>选择缩放系数: </span><br/>
    <input type = "radio" value = "1.2" name = "ratio" checked/> 1.2
    <input type = "radio" value = "1.5" name = "ratio" /> 1.5<br />
    <input type = "button" value = "+" id = "btnLarge" />
    <input type = "button" value = "-" id = "btnSmall" />
    <script src = "scripts/jquery-3.1.1.min.js"></script>
    <script>
        $(function() {
            var $div = $("#mydiv");
            $("#btnLarge").click(function() {
                changeDiv(true);
            });
            $("#btnSmall").click(function() {
                changeDiv(false);
            });
            function changeDiv(large) {
                var ratio = $("input:checked").val();
                var width = $div.css("width");
                var height = $div.css("height");
                if(large) {
                    $div.css({
                        "width": parseInt(width) * ratio + "px",
                        "height": parseInt(height) * ratio + "px"
                    });
                }
                else{
                    $div.css({
                        "width": parseInt(width) / ratio + "px",
                        "height": parseInt(height) / ratio + "px"
                    });
                }
            }
        });
    </script>
</body>
</html>
```

```
        });  
    }  
    });  
</script>  
</body>  
</html>
```

运行效果如图 6-9 所示,在这个例子中,就无法事先制作好固定的 CSS 类样式来调用,这时可以在单击按钮时,使用 jQuery 的 `css()` 方法设置 `div` 的 `style` 属性,对 `div` 根据所选比例进行比例缩放。

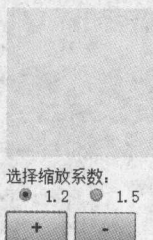


图 6-9 jQuery 非固定样式修改

6.9 jQuery 的扩展

虽然 jQuery 提供的方法已经非常方便,但有时还需要将编写好的代码封装起来提供给别人使用,例如开发自己的插件等,这就需要对 jQuery 的功能进行扩展。jQuery 的扩展方法有两种方式,一种是 jQuery 全局函数的扩展,另一种是 jQuery 对象的扩展,下面分别讨论。

1. jQuery 全局函数的扩展

jQuery 全局函数的扩展为 jQuery 的全局函数添加相应的功能,以便于随时都能方便调用,它的语法形式采用:

```
$.extend({  
    methodName:function([args]){...} //args 为可选参数  
});  
  
$.methodName([args]); //扩展后的全局调用
```

【例 6-16】 jQuery 全局函数功能扩展,实现对数组元素求和 `sum`,代码如下:

```
<!DOCTYPE html>  
<html>  
  <head>
```



```

<meta charset="utf-8">
<meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
<title>jQuery 全局函数扩展</title>
</head>
<body>
<script src="scripts/jquery-3.1.1.min.js"></script>
<script>
//为jQuery定义全局函数功能 sum,实现数组求和
$.extend({
    sum:function(array){
        var arraySum = 0;
        for(var i = 0;i<array.length;i++){
            arraySum += array[i];
        }
        return arraySum;
    }
});
var myArray = [1,3,5,7,9];
alert( $.sum(myArray));
</script>
</body>
</html>

```

2. jQuery 对象的扩展

jQuery 对象的扩展主要应用在插件开发上,为使用 jQuery 选择器查找到的对象作相应的方法扩充,它的语法形式采用:

```

$.fn.extend({
    methodName:funciton([args]){...}    //args 为可选参数
});

$(selector).methodName([args]);        //扩展后的插件调用

```

【例 6-17】 jQuery 对象的扩展,实现对复选框的选中和取消选中功能,代码如下:

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
<title>jQuery 对象的扩展</title>
<style>
    input[type='checkbox'] {
        width: 30px;
        height: 30px;
    }

```



```
    }  
    label {  
        font-size: 20px;  
    }  
    input[type='button'] {  
        margin-top: 30px;  
        width: 100px;  
        height: 30px;  
        font-size: 15px;  
    }  
}  
</style>  
</head>  
<body>  
    <label><input type="checkbox"/> HTML5 </label>  
    <label><input type="checkbox"/> JavaScript </label><br />  
    <input type="button" value="选中" id="btnCheck"/>  
    <input type="button" value="取消" id="btnUnCheck"/>  
    <script src="scripts/jquery-3.1.1.min.js"></script>  
    <script>  
        $.fn.extend({  
            check: function() {  
                $(this).each(function() {  
                    this.checked = true;  
                });  
            },  
            uncheck: function() {  
                $(this).each(function() {  
                    this.checked = false;  
                });  
            }  
        });  
        $("#btnCheck").click(function(){  
            $(".checkbox").check();  
        });  
        $("#btnUnCheck").click(function(){  
            $(".checkbox").uncheck();  
        });  
    </script>  
</body>  
</html>
```

运行效果如图 6-10 所示,这个例子中,利用 jQuery 对象的扩展方法为复选框添加了 check 和 uncheck 两个方法,可以很方便地控制复选框的选中或取消选中。

☒ HTML5 ☒ JavaScript

选中

取消

图 6-10 jQuery 对象方法扩展

6.10 jQuery 插件应用介绍

在追求页面互动效果的时代,大家都想把页面效果做得美轮美奂,这一切都离不开前端技术 JavaScript 脚本。所谓 jQuery 插件,就是由各程序员在 jQuery 的基础上开发出来,提供了良好的封装、可以直接使用的 jQuery 工具。让程序员们着迷的 jQuery 拥有丰富的插件,使得以前需要用很复杂的 JavaScript 语法以及较长时间实现的效果,在短时间内就能用便捷的语法轻松实现。

jQuery 插件的应用方式基本都是一致的,在这里以一个图片幻灯片插件 Nivo Slider 为例,简单介绍在页面中如何使用插件。

Nivo Slider (<https://github.com/gilbitron/Nivo-Slider>)号称世界最棒的轻量级 jQuery 图片幻灯插件,称它“以漂亮和易于使用而闻名于世”!它还支持响应式布局,并且完全免费开源。

从相应网址可以下载到 Nivo Slider 的文件包 Nivo-Slider-master.zip 文件,解压后的目录结构如图 6-11 所示,其中 jquery.nivo.slider.js 就是 Nivo Slider 的插件类库,可以用任意编辑器打开并查看它的源码,jquery.nivo.slider.pack.js 则是这个插件对应的压缩版(对 JS 内容作了压缩处理)。demo 目录中存放了该插件的演示页面,进入之后单击 demo.html 页面,可以看到它的运行效果,如图 6-12 所示。

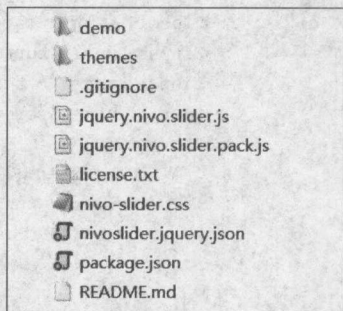


图 6-11 Nivo-Slider-master.zip 解压后的目录结构

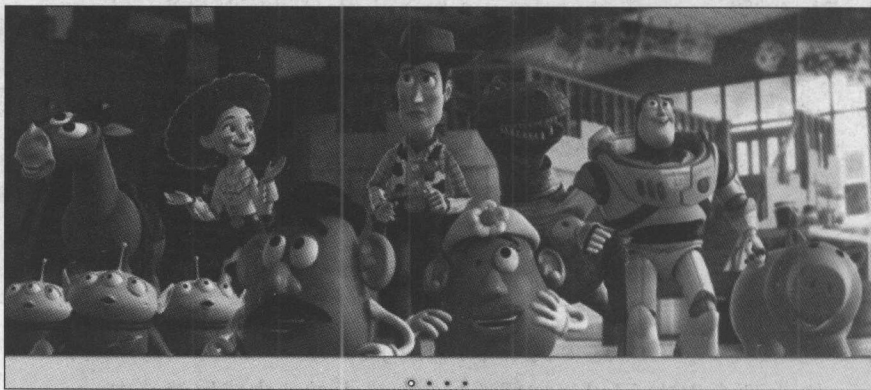



图 6-12 Nivo Slider 运行效果

这个 Demo 显示了当前流行的图片轮播效果,轮播的图片有 4 张,它们会按照幻灯效果按一定的时间间隔轮播显示,在显示过程中,图片切入方式有不同的特效,用户可以单击左右箭头实现图片的手工切换。第一张图片和第三张只是单纯的图片显示;第二张图片实现了相应的图片超链接,图片上有单独的纯文字说明;第四张的说明是一段 HTML 文本,其中有个超链接。下面尝试分析这个插件的 demo,简单说明插件的使用方法要点。

【例 6-18】 jQuery 插件 Nivo Slider 应用使用分析如下:

(1) 在页面要引入相应的 .js 文件: 先引入 jQuery 类库后, 再引入 Nivo Slider 插件类库(jquery.nivo.slider.js 或 jquery.nivo.slider.pack.js), 是否引入压缩版本取决于你是否需要对插件进行相应的调试。一般来讲, 这些插件都比较成熟, 直接使用压缩版本就可以了, 下面是引入的示例:

```
<script src = "../scripts/jquery-3.1.1.min.js"></script>
<script src = "../scripts/jquery.nivo.slider.pack.js"></script>
```

 成熟的插件一般可以使用最新版本的 jQuery 类库, 尽量避免一个项目中多个版本的 jQuery 类库。

(2) 插件的应用中, HTML 代码结构基本都是固定的, 插件的开发也是基于这部分 HTML 代码的, 所以必须按插件的 demo 中的结构书写 HTML 代码, Nivo Slider 的 HTML 页面代码结构如下:

```
<div class = "slider-wrApper theme-default">
  <div id = "slider" class = "nivoSlider">
    <img src = "images/toystory.jpg" data-thumb = "images/toystory.jpg" alt = "" />
    <a href = "http://dev7studios.com">
      <img src = "images/up.jpg" data-thumb = "images/up.jpg"
alt = "" title = "This is an example of a caption" />
    </a>
    <img src = "images/walle.jpg" data-thumb = "images/walle.jpg"
alt = "" data-transition = "slideInLeft" />
    <img src = "images/nemo.jpg" data-thumb = "images/nemo.jpg" alt = "" title = "#
htmlcaption" />
  </div>
  <div id = "htmlcaption" class = "nivo-html-caption">
    <strong>This</strong> is an example of a <em>HTML</em> caption with <a href = "#>a link</a>.
  </div>
</div>

  <img src = "images/up.jpg" data-thumb = "images/up.jpg"
alt = "" title = "This is an example of a caption" />
  </a>
  <img src = "images/walle.jpg" data-thumb = "images/walle.jpg"
alt = "" data-transition = "slideInLeft" />
  <img src = "images/nemo.jpg" data-thumb = "images/nemo.jpg" alt = "" title = "#
htmlcaption" />
  </div>
  <div id = "htmlcaption" class = "nivo-html-caption">
    <strong>This</strong> is an example of a <em>HTML</em> caption with <a href = "#>a link</a>.
  </div>
</div>
```


对比 demo 效果分析可以得知：①图片的显示都是使用 `img` 标签；②图片超链接采用 `a` 标签嵌套 `img` 标签；③图片的标签文字说明采用了在 `img` 标签中添加 `title` 属性制作；④如果图片的说明需要使用 HTML 内容，需要额外制作一个 `div` 作为容器，比如例子中 `id="htmlcaption"` 的 `div`，它的 `class="nivo-html-caption"`，这个属性应该是插件为了显示图片标签说明特意制作的样式，在需要应用这个 HTML 说明内容的图片标签中，使用 `title="#htmlcaption"`，实现自动关联；⑤所有的轮滑内容必须位于同一个容器中，在这个 demo 中是 `id="slider"` 的 `div`，它的 `class="nivoSlider"`。

(3) 漂亮的幻灯效果是基于固定的 HTML 内容以及在此基础上设计出的 CSS 样式，在这个 demo 例子中，你会发现它链接了多个 `.css` 文件，从英文的意思以及相应的目录查看，基本可以判断出，Nivo Slider 这个插件还设计出了 `default`、`bar`、`dark`、`light` 四个不同的风格，在这个 demo 中，为了使用指定的风格，它在最外面设计出了一个 `div` 容器，皮肤的指定是依靠它使用 `class="slider-wrApper theme-default"`，这里可以把 `defalut` 修改成其他 3 个值以查看不同的效果。当然，不是所有的插件都设计有不同的风格，但在插件使用中，一定不要忘记将相应的 `.css` 文件链接到页面中。实际应用中也只需要一个风格的 `.css` 文件。

(4) 使用插件提供的方法。插件的封装一般比较完善，实现时也只需要简短的语句，在这个 demo 中，使用的方法如下：

```
$('#slider').nivoSlider();
```

当然，方法调用时可能还需要一些参数配置，这需要仔细阅读插件的 API 文档以及查看它的 demo 演示。

6.11 实例：记忆翻牌游戏

【例 6-19】 使用 jQuery 和 CSS3 完成一个记忆翻牌游戏，游戏效果如图 6-13 所示。

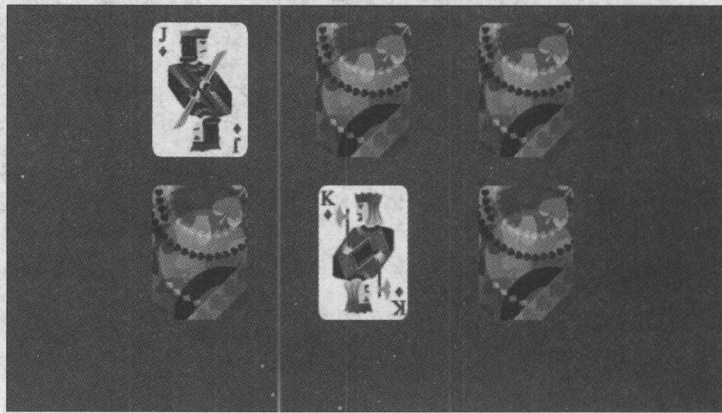


图 6-13 游戏效果

这个例子是我们比较熟悉的“记忆翻牌游戏”，桌面上有 6 张牌，你可以任翻 2 张牌，如果它们不相同，则牌都会翻回去，如果相同，则牌会消失，当所有牌都消失时，游戏结束，程序

会提示你花费的时间,你也可以选择再玩一次,如图 6-14 所示。

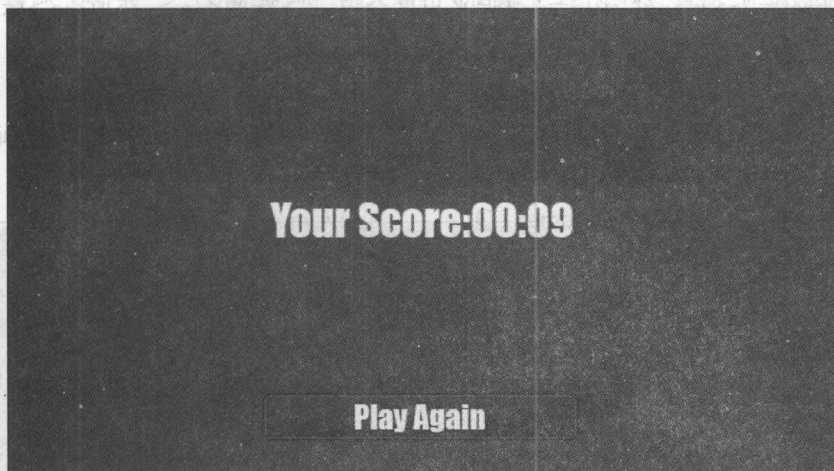


图 6-14 游戏结束

在这个例子中,演示了 jQuery 的各种应用,包括 jQuery 的查找器、DOM 操作、动态修改样式、复制元素节点等知识,翻牌的动作则是应用了 CSS 的过渡和变换实现了动画效果。

小结

本章主要讲解了 jQuery 的编程基础,介绍了 jQuery 脚本库在页面中的使用、有特色的自定义选择器,并讲解了 jQuery 对象与 DOM 对象的转换,jQuery 页面载入事件,jQuery 的事件监听、遍历方法,jQuery 在 DOM 交互上的一些操作,也介绍了如何对 jQuery 实现扩展,并以 Nivo Slider 插件为例,归纳了插件使用需要注意的一些地方。本章所讲解的内容都是 jQuery 的核心编程内容,其他内容请自行参看 API 手册,掌握这些内容之后,在实际的项目开发中能带来事半功倍的效果。

习题

一、选择题

- jQuery 中使用()可选取 id 为 div1 的元素。
A. \$ div1 B. \$("div1") C. \$(div1) D. \$("#div1")
- jQuery 中使用()表示 HTML 文档对象。
A. \$ document B. document
C. \$(document) D. this-> document
- 在 jQuery 中指定一个样式类,如果存在就自动移除,不存在就自动添加。下面的()方法可以完成这项功能。
A. removeClass() B. addClass() C. toggleClass() D. setClass()
- 下面的()选项是用来追加到指定元素的末尾的。

A. insertAfter() B. prepend() C. AppendTo() D. after()

5. 新闻, 获取<a>标签 title 的属性值是()。

A. \$("a").attr("title").val(); B. \$("#a").attr("title");
C. \$("a").attr("title"); D. \$("a").attr("title").value;

6. 执行下面代码, 单击按钮后, 会有()效果。

```
<input type="button" id="mybutton" value="单击我"/>
<script>
    $(function(){
        $("#mybutton").click(function(){
            alert("click second time");
        });
        $("#mybutton").click(function(){
            alert("click first time");
        });
    })
</script>
```

A. 弹出一对话框, 显示“click first time”
B. 弹出一对话框, 显示“click second time”
C. 弹出两次对话框, 依次显示“click first time”和“click second time”
D. 弹出两次对话框, 依次显示“click second time”和“click fist time”

7. 页面中有个 select 标签, 代码如下, 要使“选项 3”被选中的 jQuery 正确写法是()。

```
<select id="sel">
    <option value="0"> 选项 1</option>
    <option value="1"> 选项 2</option>
    <option value="2"> 选项 3</option>
</select>
```

A. \$("#sel").val("选项 3")
B. \$("#sel").val("4")
C. \$("#sel > option:eq(2)").checked
D. \$("#sel option:eq(2)").attr("selected")

8. 页面中有个 ul 标签, 代码如下, 下面()选项是错误的。

```
<ul>
    <li title='Applie'>苹果</li>
    <li title='orange'>橘子</li>
    <li title='banana'>香蕉</li>
</ul>
```


- A. `var $li = $("<li title='pear'>梨子");`是创建节点
 B. `$("#ul").Append($("#<li title='pear'>梨子"));`是给 ul 追加节点
 C. `$("#ul li:eq(1)").remove();`是删除 ul 下“橘子”那个节点
 D. 以上说法都不正确

9. 在 jQuery 中,如果想要从 DOM 中删除所有匹配的元素,下面()选项是正确的。

- A. `delete()` B. `remove()` C. `removeAll()` D. `empty()`

10. 在 jQuery 中,检查

- A. `if($("#myinput")){...}`
 B. `if($("#myinput").length>0){...}`
 C. `if($("#myinput").size>0){...}`
 D. `if($("#myinput").length()>0){...}`

二、判断题

1. 如需使用 jQuery,无需做任何事情。大多数浏览器都内置了 jQuery 库。()
 2. jQuery 的选择器符号是%。()
 3. jQuery 的选择器兼容 CSS3 选择器。()
 4. jQuery 使用 `style()`方法来设置 HTML 标签的 `style`属性。()
 5. `$("div")`是选取页面中的第一个 div 元素。()

三、填空题

1. 在一个表单中,如果要选取所有被选中的 input 标签,可以用 jQuery 的_____实现。

2. jQuery 中实现遍历的方法是_____。
 3. 要修改 HTML 元素标签的内部文字,可以使用 jQuery 的_____属性。
 4. 要把 DOM 对象转换成 jQuery 对象,可以使用 jQuery 的_____方法进行封装。
 5. jQuery 中的_____方法可以让 HTML 元素对象监听的事件只执行一次。
 6. 对 jQuery 的全局函数实现扩展,需要使用_____方法。
 7. jQuery 复制 HTML 元素时,如果需要同时复制元素的事件,要传入参数_____。
 8. 在 jQuery 中把 HTML 代码附加到被选节点内部结头部的的方法_____。

四、简答题

1. jQuery 的 `$(document).ready`方法与 JavaScript 中的 `window.onload`事件有何不同?

2. jQuery 对象与 JavaScript 的 DOM 对象如何实现转换?

五、编程题

学生成绩数据如图 6-15 所示,请使用 jQuery 完成下列工作:

- (1) 增加一列“平均成绩”,显示每个学生的三门课的平均成绩,保留一位小数点;
 (2) 把身为党员的,并且平均成绩高于 87 分的女同学数据行使用红色背景标注。

AJAX 由 4 种主要技术集合而成,每种技术在其中都有一定的职责,如表 7-1 所示。

学号	姓名	性别	党员	高等数学	大学英语	计算机文化基础
13310320712	陈中华	男	<input checked="" type="checkbox"/>	87	83	94
13310320713	王楠	女	<input checked="" type="checkbox"/>	84	85	93
13310320714	杨佳敏	女	<input type="checkbox"/>	88	89	96
13310320715	李茂杨	男	<input checked="" type="checkbox"/>	82	84	93
13310320716	赵家伟	男	<input type="checkbox"/>	79	82	90
13310320717	张思琪	女	<input checked="" type="checkbox"/>	94	82	90

图 6-15 学生数据

第7章

CHAPTER 7

AJAX 通信技术

学习目标

- 了解 AJAX 通信技术。
- 掌握 Fiddler 工具的使用,并通过它熟练掌握 HTTP 协议。
- 掌握 AJAX 核心对象 XMLHttpRequest 的使用。
- 掌握使用 FormData 对象。
- 了解 RESTful API 的概念及使用。
- 掌握 jQuery 中实现 AJAX 通信的各种方法。

和传统网页开发不同,HTML5 App 手机客户端与服务器端的数据交互不再依靠 Form 表单,而是借助于 AJAX 通信技术。本章将针对 HTML5 App 移动应用开发中常用的与服务器进行数据交互的 AJAX 通信技术作详细讲解。

7.1 AJAX 技术介绍

AJAX(Asynchronous JavaScript and XML,异步 JavaScript 和 XML)极大地发掘了 Web 应用程序的潜力,开启了大量新的可能性,缩短了 Web 程序与 Windows 程序在可用性上的差距。AJAX 应用程序的优势在于:

- 通过异步模式,提升了用户体验;
- 优化了浏览器和服务器之间的传输,减少不必要的数据往返,减少了带宽占用;
- AJAX 引擎在客户端运行,承担了一部分本来由服务器承担的工作,从而减少了大用户量下的服务器负载。

传统的 Web 开发技术中,浏览器和服务器之间的通信和数据交互主要依赖于 Form 表单,这不可避免地带来整个页面的刷新。图 7-1 和图 7-2 所展示的两个常见的 Web 效果都是 AJAX 技术的典型应用,可以想象,如果这两个效果采用传统的 Form 表单交互,页面不断地刷新,用户的体验会有多糟糕,而 AJAX 通信技术可以实现页面局部刷新,就是能在不更新整个页面的前提下维护数据,这使得 Web 应用程序更为迅捷地回应用户动作,并避免了在网络上发送那些没有改变过的信息,这对于 HTML5 App 的开发尤为重要,App 的用户体验要求较高,利用 AJAX 技术,可以避免页面切换的白屏,提高页面的快速反应能力。

AJAX 由 4 种主要技术集合而成,每种技术在其中都有一定的职责,如表 7-1 所示。

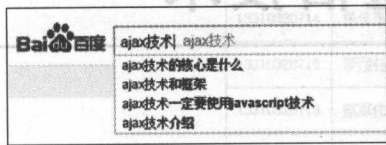


图 7-1 Baidu 的输入智能提示效果

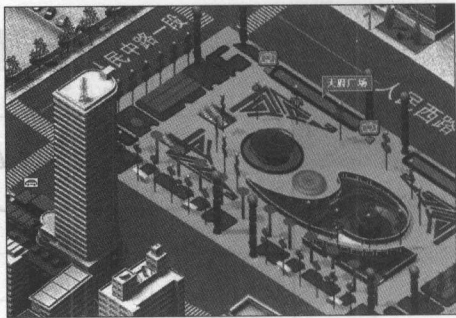


图 7-2 E 都市的三维电子地图效果

表 7-1 AJAX 的技术组成

名 称	职 责
JavaScript	通用的脚本语言,嵌入到 Web 应用中。浏览器自带的解释器允许通过与浏览器的很多内建功能进行交互。AJAX 程序都是由 JavaScript 完成的
CSS	CSS 为页面提供一种可重用的可视化样式的定义方法,它提供简单而强大的方法,以一致的方式定义和使用可视化样式。在 AJAX 应用程序中,用户界面的样式通过 CSS 来独立修改
DOM	使用 JavaScript 来修改页面,AJAX 程序在页面运行时动态修改页面,或高效重绘页面中的某个部分
XMLHttpRequest 对象	XMLHttpRequest 对象允许 Web 程序以异步方式获取数据,数据格式目前流行使用 XML 或 JSON,也可以支持其他文本格式。目前所有的浏览器都支持这个对象

从表 7-1 可以看出,AJAX 并不是一种全新的技术,它仅仅是“新瓶装老酒”,对于一些基于 Web 标准的传统技术的重新包装,是对传统技术的发展和增值。图 7-3 展示了 AJAX 技术的工作原理,JavaScript 在其中就像“胶水”一样,把各技术黏合在了一起。

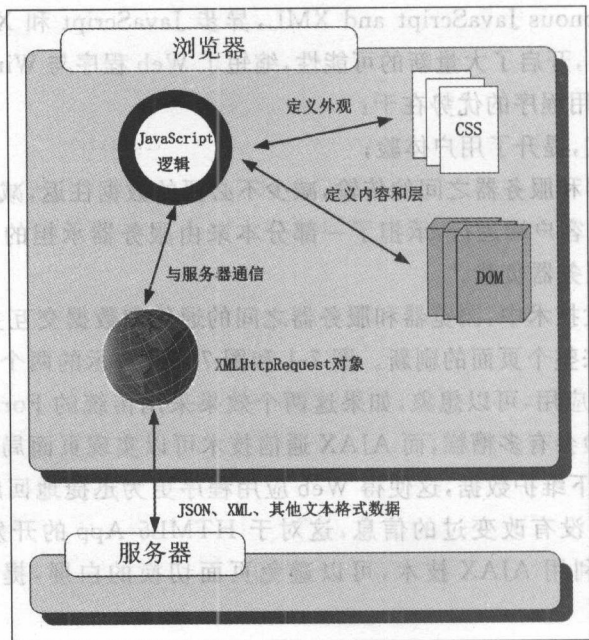


图 7-3 AJAX 工作原理

7.2 HTTP 协议分析

协议是指计算机通信网络中两台计算机之间进行通信时所必须共同遵守的规定或规则。HTTP 协议(HyperText Transfer Protocol, 超文本传输协议)是互联网上应用最为广泛的一种网络协议。AJAX 技术的核心思想其实就是使用 XMLHttpRequest 对象模拟传统的 form 表单与服务器进行数据交互,这就要求必须熟练掌握 HTTP 协议。只要牢牢掌握了 HTTP 协议,在 HTML5 App 的数据交互编程中就会无往而不利。

7.2.1 HTTP 协议介绍

HTTP 协议用于从 www 服务器将超文本传输到本地浏览器的传输协议。它可以使浏览器更加高效,使网络传输减少。它不仅保证计算机正确快速地传输超文本文档,还能确定传输文档中的哪一部分,以及哪部分内容首先显示(如文本先于图形)等。HTTP 协议是客户端浏览器或其他程序与 Web 服务器之间的应用层通信协议。在 Internet 上的 Web 服务器上存放的都是超文本信息,客户机需要通过 HTTP 协议传输所要访问的超文本信息。HTTP 包含命令和传输信息,不仅可用于 Web 访问,也可以用于其他因特网/内联网应用系统之间的通信,从而实现各类应用资源超媒体访问的集成。

HTTP 协议是由请求和响应两部分构成,一个请求对应于一个响应,称为一次会话,这是标准的客户端服务器模型。它是无状态协议,无状态是指不能进行用户状态的跟踪,也就是说,在客户端与服务器的请求和响应结束后,在服务器上并不保存任何客户端的信息。下面将使用工具 Fiddler 来熟悉 HTTP 协议。

7.2.2 Fiddler 抓包神器

Fiddler(详见 <http://www.telerik.com/fiddler>)是一个功能相当强大的 Web 调试工具,目前的最新版本是 4.6.3,它是一个 C# 实现的抓包和调试工具。Fiddler 启动后作为一个 Proxy(代理)存在于客户端和服务端之间,从中监测客户端与服务端之间的 HTTP/HTTPS 级别的网络交互,目前可以支持各种主要浏览器,例如 IE、Chrome、Firefox、Safari、Opera。Fiddler 的工作原理如图 7-4 所示。

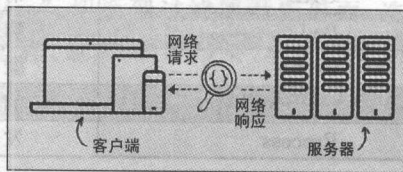


图 7-4 Fiddler 工作原理图

Fiddler 能记录客户端和服务器的所有 HTTP 和 HTTPS 请求,允许用户监视,设置断点,甚至修改输入输出数据。它还可以用来监控手机与服务端之间的 HTTP/HTTPS 通信,是开发 HTML5 App 的重要调试工具,Fiddler 启动后的界面如图 7-5 所示。

Fiddler 的主界面主要包括 4 个部分:

- 菜单栏: 包括配置、会话保存成压缩包、载入压缩包、各种浏览器的 User-Agent 模拟、设置捕获规则等。
- 工具栏: 针对当前 session(会话)的一些操作,如暂停、删除、清除缓存等。
- session 列表: 界面左边会列出 Fiddler 抓取到的每次 HTTP 会话(每一条称为一个

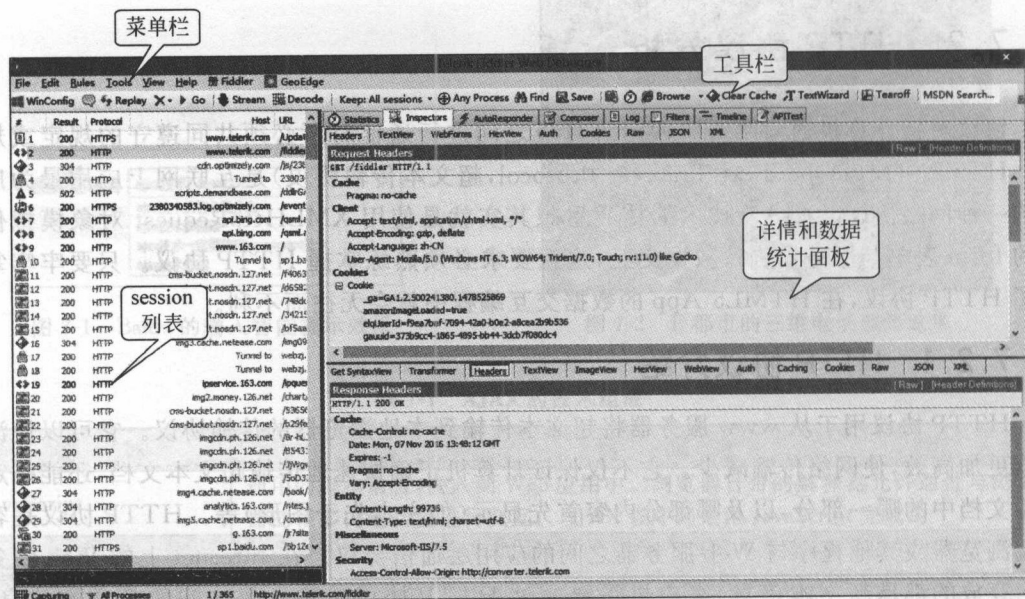


图 7-5 Fiddler 启动后的界面


session), 主要包含了会话的编号、URL 地址、响应状态码、所用协议等信息, session 列表可以使用鼠标左键控制宽度显示, 标题上的字段含义如表 7-2 所示。

表 7-2 session 标题各字段含义

名 称	举 例
#	HTTP 会话编号, 从 1 开始, 按 HTTP 的请求顺序递增
Result	HTTP 的响应状态码
Protocol	HTTP 或 HTTPS
Host	请求地址的域名
URL	请求的服务器路径和文件名
Body	响应内容主体的字节数
Cache	缓存或过期的时间控制
Content-Type	响应消息的类型
Process	发出此请求的 Windows 进程及进程 ID

- 详情和数据统计面板: 界面右边是针对每条会话的一些具体统计(例如发送/接受字节数、发送/接收时间)和数据包分析等。

启动 Fiddler 后会发现, 在 session 列表中会不断出现 HTTP 会话, 这是因为机器上的某些软件在后台会不断向服务器发送某些数据(有些时候用户的隐私就是这样被窃取的), 你可以鼠标左键单击某个 session, 按 Delete 键进行删除, 也可以按 Ctrl+A 组合键选中所有的 session, 按 Delete 键进行全部删除。

Fiddler 界面的左下角有个图标 , 表示 Fiddler 目前处于抓取状态, 可以用鼠标左键单击关闭抓取, 图标会消失, 再次单击会再次恢复抓取。

在会话列表中, 为了方便查看, 可以用鼠标左键选中某个会话, 单击鼠标右键, 选择

“Mark”菜单中的颜色,对会话进行标注,如图 7-6 所示,编号 4 和 24 的会话分别以红色和蓝色进行了标注(编辑注:由于印刷的原因,图中仅以不同灰度以示区分)。

#	Result	Protocol	Host	URL
3	200	HTTP	edick.baidu.com	/a.js?tu=u27
4	200	HTTP	www.163.com	/
5	200	HTTP	web.stat.ws.126.net	/stat/projec
6	200	HTTP	pagead2.googlesyndication.com	/activeview?
7	204	HTTP	pagead2.googlesyndication.com	/pagead/gen
24	200	HTTP	ipservice.163.com	/ipquery

图 7-6 标记会话

为了方便以后查看和分析,不希望每次都使用 Fiddler 抓取,可以先选中会话,如图 7-7 所示,单击鼠标右键,选择“Save”菜单,再选择“Selected Sessions”菜单,选中“in ArchiveZIP”,把选中的会话保存成 .saz 的文件,需要再次查看时,可以直接双击该文件导入 Fiddler 进行查看。

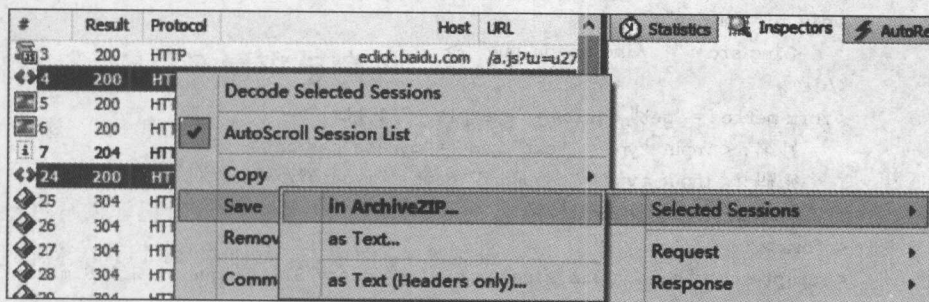


图 7-7 保存会话

由于 Fiddler 会抓取本机的所有程序的 HTTP 请求,为了更清晰地查看结果,我们经常希望 Fiddler 只抓取指定的 HTTP 会话,这需要使用相应的过滤器功能。如图 7-8 所示,在“详情和数据统计面板”中,鼠标左键单击 Tab 项“Filters”,切换到过滤器设置界面,选中“Use Filters”选项后,可以根据需求指定主机名、系统进程、请求/响应报头、响应码等的各种过滤功能。设置好后,如图 7-9 所示,用鼠标左键单击 Actions 按钮,单击 Run Filterset now 便可以立即实现相应的过滤。

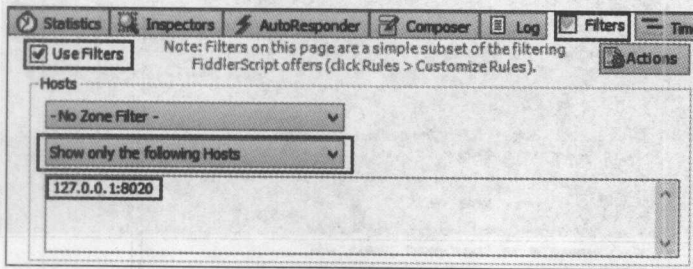


图 7-8 过滤器设置界面

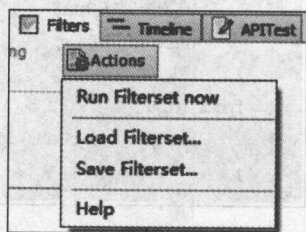


图 7-9 执行过滤



Fiddler 有时在使用时设置了过滤器后忘了关闭,会造成无法抓取其

他 HTTP 请求数据, 可以通过“Filters”前面的复选框选中与否知道是否打开了过滤器, 也可以对图 7-8 中的“Use Filters”选项进行关闭。

下面将用一个简单的例子说明 Fiddler 工具的使用, 用它来抓取 HTTP 请求, 再用它来熟悉 HTTP 协议。

【例 7-1】 Fiddler 抓取 HTTP 会话使用示例, 方法如下:

(1) 设计好相应的测试网页, 相应的页面代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Fiddler 抓取 HTTP 请求示例</title>
    <link rel="stylesheet" href="css/example-7.1.css" />
  </head>
  <body>
    <div id="mydiv">
      
    </div>
    <form method="get" action="example-7.1.html">
      姓名: <input type="text" name="myname"/><br />
      密码: <input type="password" name="mypass"/><br />
      <input type="submit"/>
    </form>
    <script src="http://lib.sinaApp.com/js/jquery/3.1.0/jquery-3.1.0.min.js">
    </script>
  </body>
</html>
```

(2) 所用到的. css 文件设计内容如下:

```
body{
  margin: 0 auto;
  text-align: center;
}
img{
  width: 180px;
  height: 60px;
}
form input{
  margin-bottom: 15px;
}
```

(3) 打开 Fiddler, 配置相应的过滤器, 如图 7-8 所示, 对“Hosts”进行过滤, 在下拉框中选择“Show only the following Hosts”后, 在文本框中输入“127. 0. 0. 1: 8020”(这是 HBuilder 中自带 Web 服务器运行页面的 URL 和端口号)。

(4) 在 HBuilder 中启动测试页面后在 Chrome 中浏览, 效果如图 7-10 所示。

(5) 切换到 Fiddler 会发现,虽然只有一张页面,但实际上这张页面一共有 3 次请求,首先是对页面 .html 的请求,接着是 .css,最后是图片,在会话列表中使用不同的图标进行了标示,如图 7-11 所示。

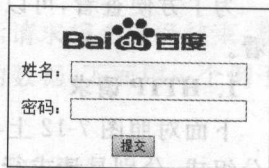


图 7-10 在 Chrome 中浏览的界面

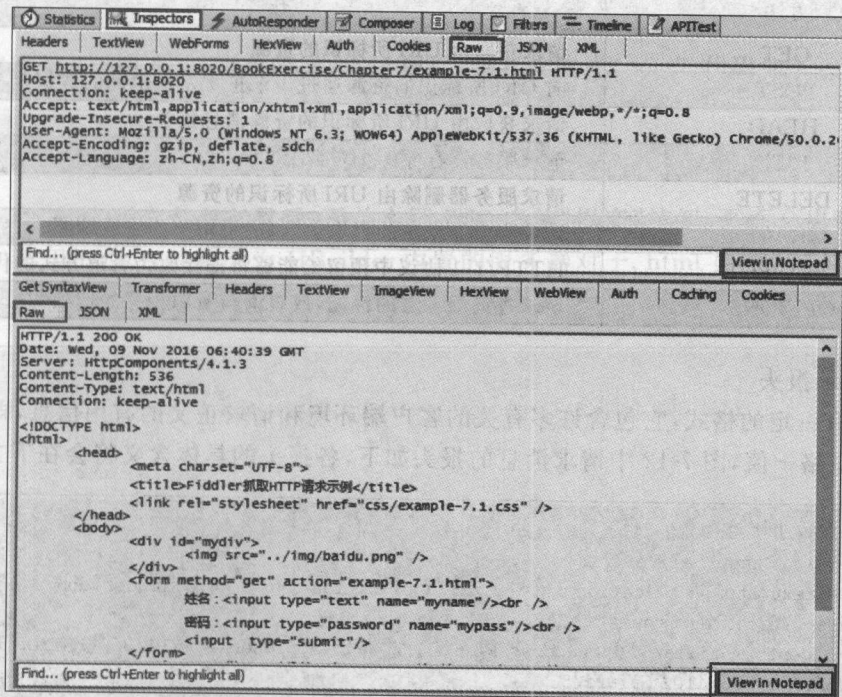
HTTP 协议是基于请求/响应的协议,页面以及页面所引用的资源(图片、音频、CSS 文件、js 文件)都是浏览器根据解析结果向服务器发送新的请求得到的。所以 Web 前端性能优化的第一条原则就是——尽量减少页面的 HTTP 请求次数。

#	Result	Protocol	Host	URL	Body	Caching	Content-Type
6	200	HTTP	127.0.0.1:8020	/BookExercise/Chapter7/example-7.2.html	536		text/html
7	200	HTTP	127.0.0.1:8020	/BookExercise/Chapter7/css/example-7.2.css	130		text/css
8	200	HTTP	127.0.0.1:8020	/BookExercise/img/baidu.png	3,706		image/png

图 7-11 测试页面的 HTTP 请求

7.2.3 HTTP 请求与响应

在如图 7-11 所示的 Fiddler 所抓取请求中,单击鼠标左键选中编号为“6”的 .html 的请求,在右边“详情和数据统计面板”中,选中 Tab 项“Inspectors”后,再把出现的上下两个面板的 Tab 选项的“Raw”都选中,这时上下面板出现的内容就是一个 HTTP 会话对应的 HTTP 请求和 HTTP 响应内容,如图 7-12 所示。



上半部分——请求消息;下半部分——响应消息

图 7-12 页面的 HTTP 请求与响应

为了方便查看,可以鼠标单击每个面板中的“View in Notepad”按钮,用“记事本”软件查看。

1. HTTP 请求

下面对照图 7-12 上半部分的 HTTP 请求的内容来讲解 HTTP 协议。HTTP 请求由 3 部分组成,分别是请求行、请求报头、请求正文。

1) 请求行

请求消息的第一行就是请求行,它的格式是:

```
Method Request - URI HTTP - Version CRLF
```

- Method: HTTP 请求的方法;
- Request-URI(Uniform Resource Identifier,统一资源标识符): 请求的资源地址;
- HTTP-Version: 协议的版本;
- CRLF: 代表回车。

在图 7-12 中,HTTP 请求行的内容为:

```
GET http://127.0.0.1:8020/BookExercise/Chapter7/example-7.2.html HTTP/1.1
```

它表示对“http://127.0.0.1:8020/BookExercise/Chapter7/example-7.2.html”发出了 GET 请求,协议版本是 HTTP 1.1。常用的 HTTP 请求方法及作用见表 7-3。

表 7-3 常用的 HTTP 请求方法

方法	作 用
GET	请求获取由 URI 所标识的资源
POST	向 URI 所标识的资源通过请求正文发送附加的数据
HEAD	请求获取由 URI 所标识的资源消息报头
PUT	请求服务器存储一个资源
DELETE	请求服务器删除由 URI 所标识的资源
TRACE	请求服务器回送的请求信息,主要用于测试或诊断
CONNECT	HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务
OPTIONS	请求查询服务器的性能,或查询与资源相关的选项和需求

2) 请求报头

报头有一定的格式,它包含许多有关的客户端环境和请求正文的有用信息,即报头名字+“:”+空格+值,图 7-12 中请求消息的报头如下,各报头的具体含义将会在下文介绍。

```
Host: 127.0.0.1:8020
Connection: keep-alive
Accept:text/html,Application/xhtml+xml,Application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/50.0.2661.102 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
```

3) 请求正文

请求报头和请求正文之间是一个空行,这个行非常重要,它表示请求报头已经结束,接下来的是请求正文。请求正文主要用来包含客户端向服务端发送的数据,下面用一个例子来解释请求正文。

【例 7-2】 HTTP POST 方法的请求正文传递数据,方法如下:

(1) 例 7-1 运行之后,输入用户名和密码,再单击图 7-10 中的“提交”按钮,会再次出现页面 example-7.1.html,虽然这张页面和图 7-10 的效果完全一样,但是它已经不再是前面那张页面了。用 Fiddler 抓取消息,你会发现又抓取了 3 次请求(也就意味着这张页面被浏览器重新加载了)。

(2) 注意浏览器地址栏中显示的地址,如果 form 表单采用 get 方式提交,数据是附加在 URL 后面+“?”+键值对传递的。

URL 地址为:

```
http://127.0.0.1:8020/BookExercise/Chapter7/example-7.1.html?myname=huangbo&mypass=1234
```

用 Fiddler 抓取到的请求信息的请求行为如下:

```
GET http://127.0.0.1:8020/BookExercise/Chapter7/example-7.1.html?myname=huangbo&mypass=1234 HTTP/1.1
```

结合 form 表单的输入可知,这里 myname=huangbo&mypass=1234 是要发送给服务器的数据。

(3) 修改例 7-1 中 form 表单的 method 属性,将其修改成“post”,代码如下:

```
<form method="get" action="example-7.1.html">
```

再次运行,输入用户名和密码后,再单击图 7-10 中的“提交”按钮,页面会出现“内部服务器错误”,不用担心,这只是因为 HBuilder 内置的服务器对于.html 页面无法处理 POST 请求造成的,只需要关心它的 HTTP 请求消息:

```
POST
http://127.0.0.1:8020/BookExercise/Chapter7/example-7.1.html HTTP/1.1
Host: 127.0.0.1:8020
Connection: keep-alive
Content-Length: 26
Cache-Control: max-age=0
Accept: text/html, Application/xhtml+xml, Application/xml;q=0.9, image/webp, */*;q=0.8
Origin: http://127.0.0.1:8020
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
Content-Type: Application/x-www-form-urlencoded
```

```
Referer: http://127.0.0.1:8020/BookExercise/Chapter7/example-7.1.html
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8

myname=huangbo&mypass=1234
```

(4) 请求消息中的“myname=huangbo&mypass=1234”就是 form 表单 post 方法传递过去的数 据，它出现在请求正文中，与请求报头之间隔了一行。



对于 HTTP 协议，无论是通过 GET 方式还是 POST 方式，数据的传递都是明文传递，安全性极差，所以需要 HTTPS(Hyper Text Transfer Protocol over Secure Socket Layer)。

HTTPS 是在 HTTP 的基础上加入了 SSL(Secure Socket Layer)协议，SSL 依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。苹果公司已在 2016 年推出措施，要求未来所有的 iOS App 必须通过 HTTPS 实现通信。

2. HTTP 响应

在接受和解释 HTTP 请求消息后，服务器会返回一个对应的 HTTP 响应消息，如图 7-12 下半部分所示。与 HTTP 请求类似，HTTP 响应消息也是由 3 个部分组成，分别是：状态行、响应报头、响应正文。

1) 状态行

状态行由协议版本、数字形式的状态码以及相应的状态描述组成，各元素之间以空分隔，格式是：

```
HTTP - Version Status - Code Reason - Pharse CRLF
```

- HTTP-Vesion: 服务器 HTTP 协议的版本；
- Status-Code: 服务器的响应代码；
- Reason-Pharse: 服务器响应状态的描述；
- CRLF: 表示回车。

在图 7-12 中，HTTP 状态行的内容为：

```
HTTP/1.1 200 OK
```

作为 HTML5 程序员，对于一些常用的状态码和状态描述是需要了解的。状态码由 3 位数字组成，表示请求是否被理解或满足，而状态描述则给出了关于状态代码的简短文本描述。

状态代码的第 1 个数字状态代码定义了响应的类别，后面两位没有具体的分类。第 1 个数字有 5 种可能取值，见表 7-4。

表 7-4 状态代码的几种可能取值

可能取值	含 义
1xx	指示信息——表示请求已接收,继续处理
2xx	表示请求已被成功接收、理解、接受
3xx	重定向,要完成请求必须进一步操作
4xx	客户端错误,请求有语法错误或请求无法实现
5xx	服务器端错误,服务器未能实现合法的请求

在 AJAX 程序中经常会用到的状态代码和状态描述见表 7-5。

表 7-5 常见状态代码和状态描述

状态代码	状 态 描 述	说 明
200	OK	客户端请求成功
400	Bad Request	由于客户端请求有语法错误,不能被服务器所理解
401	Unauthorized	请求未经授权,这个状态码必须和 WWW-Authenticate 报头域一起使用
403	Forbidden	服务器收到请求,但是拒绝提供服务,服务器通常会在响应正文中给出不提供服务的原因
404	Not Found	请求的资源不存在
500	Internal Server Error	服务器发生错误,导致无法完成客户端的请求
503	Service Unavailable	服务器当前不能处理客户端的请求

2) 响应报头

和请求报头类似,响应消息在状态行下面都会出现一些响应报头。在这里会允许服务器出现一些不能放在状态行中的附加响应信息,以及服务器的信息和对 Request-URI 所标识的资源进行下一步访问的信息,图 7-12 中的响应报头如下:

```
Date: Wed, 09 Nov 2016 08:07:40 GMT
Server: HttpComponents/4.1.3
Content-Length: 537
Content-Type: text/html
Connection: keep-alive
```

3) 响应正文

与图 7-12 所抓取的 HTTP 请求对应的响应消息,响应正文和响应报头之间隔了一行,正文的内容恰好就是之前设计好的,html 页面内容。和请求正文不同的是,响应正文不会永远是文本,有时会是图片或其他格式,在 Fiddler 中查看响应正文的方式需要基于返回的资源类型作调整,如图 7-13 所示。

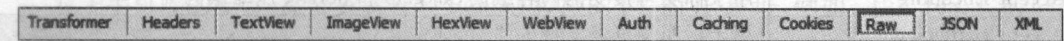


图 7-13 Fiddler 中不同格式查看的切换

如果将图 7-13 中的 Tab 项切换到 WebView,就可以看到该页面在不使用 CSS 时的响

应对应效果。对于其他的 Tab 项,简单说明如下:

- Transformer: 如果接收到的 HTTP 响应消息是经过服务器 gzip 和 deflate 压缩的,可以在这里解压查看。
- Headers: HTTP 响应消息的响应报头。
- TextView: HTTP 响应正文的内容(适合文本格式)。
- ImageView: HTTP 响应正文的内容(适合图片格式)。
- HexView: HTTP 响应正文的十六进制。
- WebView: HTTP 响应正文的在浏览器中的单独效果(适合网页格式)。
- Auth: HTTP 响应消息中的授权内容。
- Caching: HTTP 响应消息中的缓存内容。
- Cookies: HTTP 响应消息中对于 Cookie 的处理。
- Raw: 原始的 HTTP 响应消息。
- JSON: HTTP 响应正文的内容(适合 JSON 格式)。
- XML: HTTP 响应正文的内容(适合 XML 格式)。

3. 消息报头

在前面的例子中,对于 Fiddler 所抓取的 HTTP 请求以及对应的响应中,会发现有很多报头。HTTP 消息报头包括普通报头、请求报头、响应报头、实体报头。消息报头的名字与大小写无关。不要小看消息报头,浏览器的一些行为就是由报头控制的,例如 Cookie 的生成、页面的跳转等。下面就一些常见的报头作以说明,如表 7-6~表 7-9 所示。

表 7-6 常见的普通报头

报头名字	说 明
Cache-Control	用于指定缓存指令,该指令将被请求/响应链中所有的缓存机制所遵循,它会覆盖默认的缓存规则。缓存指令是单向的,在请求中出现的缓存指令,并不意味着响应中就要出现。另外,在一个消息(请求/响应消息)中指定的缓存指令,不会影响一个消息处理的缓存机制。缓存指令包括请求时的缓存指令和响应时的缓存指令。其中最常用的是 no-cache,用于指示请求或响应消息不能缓存,即: Cache-Control: no-cache
Connection	允许发送者指定连接的选项,例如指定连接是持续的,或者“close”选项,通知服务器,在响应完成后,关闭连接
Date	消息产生的日期和时间

表 7-7 常见的请求报头

报头名字	说 明
Accept	指定客户端能够处理的 MIME 类型
Accept-Charset	用于标明客户端接受的字符集,默认则可以接受任何字符集,可用来处理某些乱码的情况
Accept-Encoding	用于标明可接受的内容编码,如 gzip 和 deflate 压缩编码
Accept-Language	用于标明浏览器支持的语言,如中文
Cookie	用于向服务器发送属于该网站的 Cookie
Host	用于指定被请求资源的 Internet 主机和端口号,它通常是从 HTTP URL 中提取出来的

续表

报头名字	说 明
Upgrade-Insecure-Requests	让浏览器不再显示 https 页面中的 http 请求警报
User-Agent	标明客户端的操作系统、浏览器和其他相关信息,服务器根据这个报头判断浏览器类型,可以利用这个报头对不同的浏览器作相应的处理
Range	用于标明请求资源的字节范围,可用来实现断点续传功能

表 7-8 常见的响应报头

报头名字	说 明
Location	用于重定向到一个新的位置,常见的服务器端控制页面跳转实际就是在响应消息中用这个报头控制
Server	包含了服务器用于处理请求的 Web 服务器软件信息
Set-Cookie	服务器指示浏览器生成并存储 Cookie,如服务器端编程 Session 的 SessionId 就是靠这个报头生成的

表 7-9 常见的实体报头

报头名字	说 明
Content-Encoding	标明正文的压缩方式,客户端在使用前必须解压
Content-Language	描述资源所用的自然语言,允许用户按自身的语言来识别和区分实体
Content-Length	用于标明完全的正文长度,以十进制的字节方式表示
Content-Type	用于指明发送给浏览器的实体正文的媒体类型,浏览器会根据该报头调用不同内置引擎进行渲染
Last-Modified	用于标明资源的最后修改日期和时间
Expires	给出响应过期的日期和时间,针对浏览器的缓存,为了让其在一段时间后更新页面,可使用该报头

7.2.4 Fiddler 手机数据抓包

在移动应用开发中,程序员比较头疼的事情是真机运行时的数据抓包,Fiddler 绝对称得上是“抓包神器”,Fiddler 不但能截获各种浏览器发出的 HTTP 请求,也可以截获各种智能手机发出的 HTTP/HTTPS 请求。它的原理实现也比较简单,让智能手机与运行 Fiddler 的 PC 位于同一网段,将智能手机的网关设置为 PC 的 IP 地址,所有网络通信实际是借助于 Fiddler 这个网络代理实现通信的。

PC 端 Fiddler 的配置需要在打开 Fiddler 后,用鼠标左键单击“Tools”菜单,选中“Telerik Fiddler Options”,弹出对话框,切换 Tab 项到“Connecitons”,选中“Allow Remote Computers To Connect”,如果监听端口和其他软件或服务有冲突,需要手工修改,如图 7-14 所示。

对于手机端的配置,需要查看安装 Fiddler 程序的 PC 的 IP 地址,例如:192.168.1.6,记录下来后设置手机上网所用的代理,图 7-15 和图 7-16 分别示范了在 MIUI 8.0 和 iOS 9.3 中的代理设置。

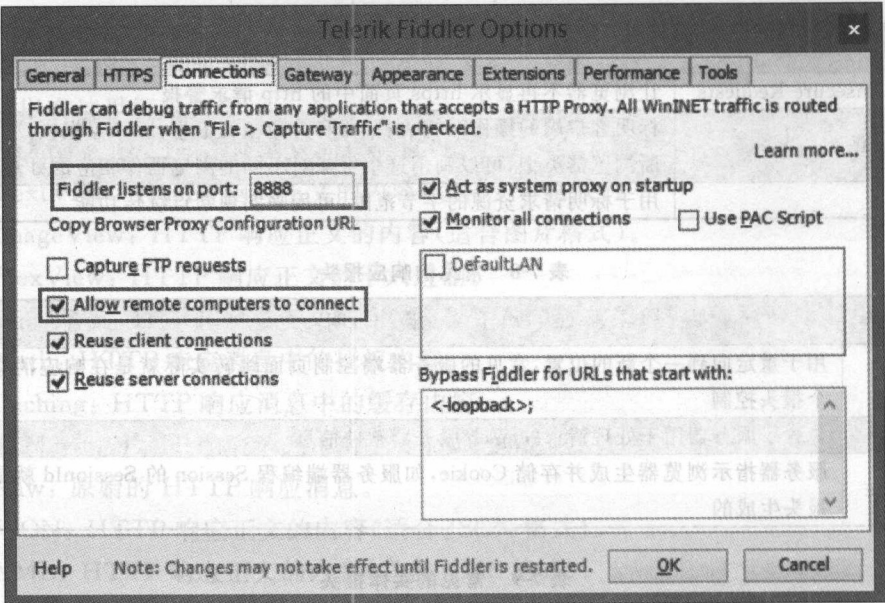


图 7-14 Fiddler 手机抓包配置



图 7-15 MIUI 8.0 代理配置

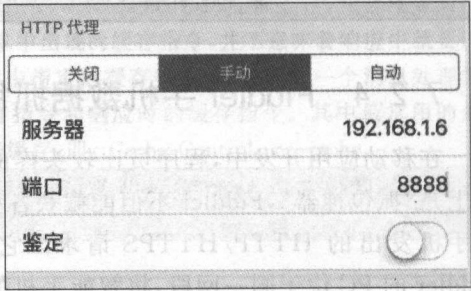


图 7-16 iOS 9.3 代理配置

7.2.5 Fiddler 模拟 HTTP 请求

在移动应用开发中,经常涉及服务器 API 通信测试,Fiddler 之所以称为“神器”,原因在于可以使用 Fiddler 很方便地模拟各种 HTTP 请求。

使用方式是打开 Fiddler 后,在“详情和数据统计面板”中切换 Tab 项到“Compose”,如图 7-16 所示,在这里可以切换请求的方法,输入 API 地址,切换 HTTP 协议版本,手工输入 HTTP 请求报头,还可以输入请求正文。这里要注意的是:请求正文如果是灰色的,则代表不能输入正文,例如 HTTP 的 GET 请求是没有正文的。构建 HTTP 请求后,用鼠标左键

单击 Execute 按钮后,可以模拟出不同的 HTTP 请求。

在图 7-17 中,有个请求报头 User-Agent,它的值是“Fiddler”,实际开发中,有可能需要使用它来模拟各种移动设备。这里以模拟 iphone6 为例,鼠标左键单击 Fiddler 的“Rules”菜单,选择“User-Agents”菜单项后,选中“iphone6”,再次模拟 HTTP 请求后,可以发现 Fiddler 模拟出了 iphone6 手机的 User-Agent:

```
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 8_3 like Mac OS X) AppleWebKit/600.1.4
(KHTML, like Gecko) Version/8.0 Mobile/12F70 Safari/600.1.4
```

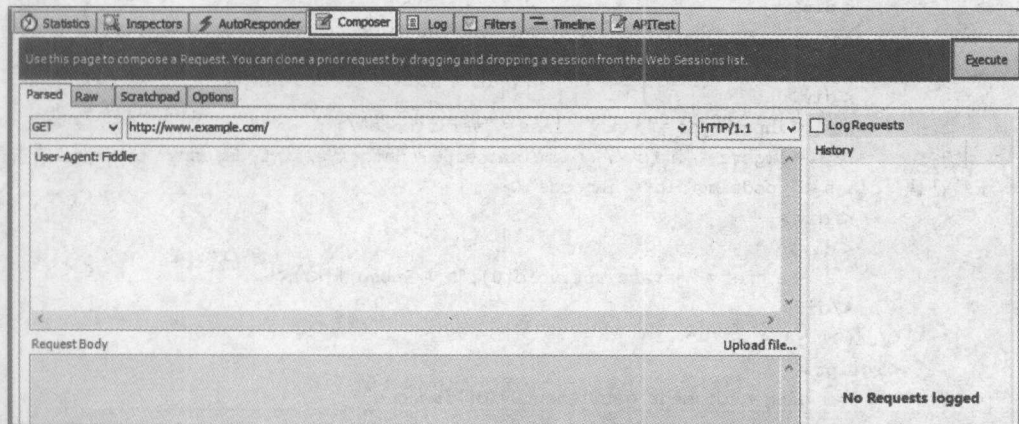


图 7-17 Fiddler 模拟 HTTP 请求

使用 Fiddler 模拟 HTTP 请求正确调用 API 时需要注意以下 4 点:

- 请求的方法: 即使对于同一个 API 采用不同的 HTTP 请求方法也可能会得到不同的结果,必须与 API 的说明完全对应。
- 请求的 URL: 要仔细阅读 API 的 URL 说明,特别是看数据是否需要 URL 传递。
- 请求的正文: 注意 API 正文格式的要求。
- 请求的报头: 这点是最容易忽略的,特别要注意某些 API 是否要求传递一些特殊的报头。

7.2.6 图片验证码

目前大多数网站在注册或登录时都会使用图片验证码,图片验证码都是随机的字符动态生成的图片,主要目的是强制人机交互来抵御机器自动化攻击,客户端可以随时单击图片验证码进行刷新,下面以客户端使用图片验证码为例。

【例 7-3】 页面使用图片验证码,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport">
```

```

content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
<title>图片验证码的使用</title>
<style>
/* CSS 代码略, 请参考源码 */
</style>
</head>
<body>
<div class = "container">
<div>
<input type = "text" placeholder = "邮箱/手机号"/>
</div>
<div>
<input type = "password" placeholder = "6-8 位密码, 区分大小写"/>
</div>
<div>
<input type = "text" class = "verifycode"/>
<imgsrc = "http://www.meishihui68.com.cn/VerifyCode.aspx" title = "单击刷新验证码" class = "vcodeimg" id = "imgcode"/>
</div>
<div>
<a href = "javascript:void(0);">注 &nbsp;  册</a>
</div>
</div>
<script>
var oimg = document.getElementById("imgcode");
oimg.onclick = function(){
this.src = "http://www.meishihui68.com.cn/VerifyCode.aspx?"
+ Math.random();
}
</script>
</body>
</html>

```

页面运行后的效果如图 7-18 所示, 每当鼠标单击图片验证码时, 图片验证码都会刷新一次。注意, 这个例子中 img 标签的 src 属性, 并不像以前会使用类似 .png、.gif 这样的图片路径(相对路径或网络的), 而是指向了一个 API 接口的 URL 地址。如果你把这个 URL 地址复制到浏览器的地址栏中, 同样也能得到一张验证码图片, 按 F5 键也可以实现刷新。

图 7-18 图片验证码

使用 Fiddler 抓取 HTTP 请求后, 每单击一次图片验证码, 就会向相应的服务器端发送一次 HTTP GET 请求, 在返回的 HTTP 响应中, HTTP 的响应消息中有一个比较重要的报头:

```
Content-Type: image/jpeg; charset = utf-8
```

这是由服务器端控制的 HTTP 响应消息输出, 当浏览器接收到这个报头时, 它会知道正在接收的消息是一张 jpg 图片, 所以 img 图片是可以正常渲染出来的。



本例中,src 属性设置为服务器端 API URL 时,必须加上随机数。这是因为有些浏览器内核会缓存 HTTP GET 请求。在 HTML5 App 与服务器端交互时也要注意避免这个问题。

7.3 XMLHttpRequest 对象

AJAX 技术中最核心的就是 XMLHttpRequest 对象,它最初的名称叫做 XMLHTTP,是微软公司为了满足 Web 开发者的需要,于 1999 年在 IE5.0 浏览器中率先推出的。后来这个技术被规范命名为 XMLHttpRequest。它正是 AJAX 技术所以与众不同的地方。简而言之,XMLHttpRequest 为运行在浏览器中的 JavaScript 脚本提供了一种在页面之内与服务器通信的手段。

目前常见的浏览器(PC 端或移动端)基本上都已经内置了这个对象。下面详细介绍这个对象在数据交互方面的使用。

7.3.1 使用方法

XMLHttpRequest 对象提供了一些常用的属性和方法,见表 7-10 和表 7-11。

表 7-10 XMLHttpRequest 对象的常用属性

属 性	说 明
onprogress	分成上传和下载两种情况:下载的 progress 事件属于 XMLHttpRequest 对象;上传的 progress 事件属于 XMLHttpRequest.upload 对象。可以设置文件上传或下载的进度处理事件
ontimeout	HTTP 请求超时事件触发器
onreadystatechange	状态改变的事件触发器,每个状态改变都会触发这个事件触发器
readyState	数值,代表 XMLHttpRequest 对象的 5 个状态
responseText	服务器的响应,字符串
responseXML	服务器的响应,XML DOM 对象
status	服务器返回的 HTTP 状态码
statusText	HTTP 状态码的相应文本
timeout	设置 HTTP 请求的时限,单位为 ms,超过时限自动停止 HTTP 请求
abort()	停止当前请求
getResponseHeader(header)	返回指定响应头的字符串值

表 7-11 XMLHttpRequest 对象的常用方法

方 法	说 明
abort()	停止当前请求
getAllResponseHeaders()	将 HTTP 请求的所有响应头作为键/值对返回
getResponseHeader(header)	返回指定响应头的字符串值
open (method, URL [, asyncFlag [, "userName" [, "password"]])	建立对服务器的请求,method 参数是 HTTP 请求方法,URL 参数可以是相对或绝对 URL。该方法还有以下 3 个可选参数。

续表

方 法	说 明
	asyncFlag: 是否非同步标记,userName: 用户名 password: 密码
setRequestHeader(header, value)	把指定请求报头设置为所提供的值,在调用该方法之前必须先调用 open 方法
send(content)	向服务器发送请求(空字符串必须是 null)

XMLHttpRequest 对象的使用可以按图 7-19 分为 5 个步骤。

1. 创建 XMLHttpRequest 对象

所有现代浏览器均支持 XMLHttpRequest 对象(IE5 和 IE6 使用 ActiveXObject),本书主要讲解 HTML5 App 开发方面的知识,所以不涉及 ActiveXObject 的用法,创建 XMLHttpRequest 对象的代码如下:

```
var xhr = new XMLHttpRequest();
```

2. 创建 HTTP 请求

在发送 HTTP 请求之前,需要调用 open()方法初始化 HTTP 请求的参数,这个方法并不真正发送 HTTP 请求。它的语法如下:

```
open(method,URL[,asyncFlag[, "userName" [, "password"]]])
```

参数说明如下:

- method: HTTP 请求方法;
- URL: 请求的 URL 地址,可以是网址或本地文件;
- asyncFlag: 布尔型,指定此请求是否为异步方式,默认为 true;
- userName,password: 是指服务器验证需要的用户名和密码,可省略。

3. 设置状态改变时的事件

XMLHttpRequest 对象有一个属性 readyState,它有 5 个值,分别对应了 5 个状态:

- 0,未初始化,对象已创建,但还未使用 open 方法;
- 1,正在加载,还未使用 send 方法;
- 2,已加载,send 方法已使用,但当前的状态未知;
- 3,交互中,接收了部分数据;
- 4,完成,数据接收完毕。

当 readyState 属性值发生改变时,XMLHttpRequest 对象会激发一个 readyStateChange 事

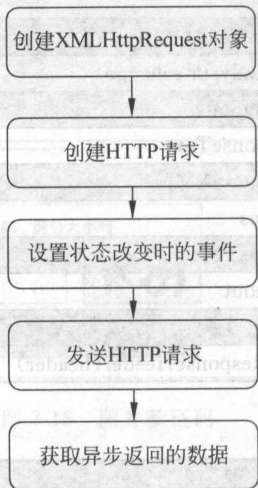


图 7-19 XMLHttpRequest 对象使用步骤

件,所以需要使用 onreadystatechange 事件来处理数据。常用的状态处理事件的代码结构如下:

```
xhr.onreadystatechange = function(){
    if(xhr.readyState == 4){//服务器已响应
        ...
    }
}
```

4. 发送 HTTP 请求

XMLHttpRequest 对象的 send() 方法负责发送 HTTP 请求,若请求中不包含请求正文,则使用:

```
xhr.send(null);
```

若请求包含正文,则将请求正文作为参数,例如:

```
xhr.send("a = 2&b = 3");
```

5. 获取异步返回的数据

XMLHttpRequest 对象在数据接收完成后,需要使用 status 或 statusText 属性,来判断请求是否成功,status 和 statusText 属性返回当前请求的响应状态代码和状态描述,这里与 HTTP 协议是一个完全对应的关系。

要取出异步返回的数据,可以使用 responseText 属性,它返回对应的就是 HTTP 响应正文,如果要对应处理 xml 数据,则需要用到 responseXML 属性,结合第 3 步,一般代码结构如下:

```
xhr.onreadystatechange = function(){
    if(xhr.readyState == 4){
        if(xhr.status == 200){
            var res = xhr.responseText;
            var xmlDom = xhr.responseXML;
        }
    }
}
```

7.3.2 读取数据

这里将用一个完整的例子展现 XMLHttpRequest 对象的使用,效果如图 7-20 所示。

【例 7-4】 网页启动时,从服务器获取并显示一个 .css 文件内容在 textarea 中,代码如下:

读取的CSS文件内容

```
body{
    margin: 0 auto;
    text-align: center;
}
img{
    width: 180px;
    height: 60px;
}
form input{
    margin-bottom: 15px;
}
```

图 7-20 XMLHttpRequest 异步读取 .css 文件内容并显示

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
    <title>XMLHttpRequest 对象读取数据</title>
    <style>
      /* CSS 代码略, 见配套源码 */
    </style>
  </head>
  <body>
    <div id="container">
      <p>
        读取的 CSS 文件内容
      </p>
      <textarea id="csscontent" cols="40" rows="15">
    </textarea>
    </div>
    <script>
      var odiv = document.getElementById("csscontent");
      //1. 创建 XMLHttpRequest 对象
      var xhr = new XMLHttpRequest();
      //2. 构建 Http 请求
      xhr.open("GET",
        "http://www.meishihui68.com.cn/css/example-7.1.css", true);
      //3. 设置状态改变时的事件
      xhr.onreadystatechange = function() {
        if(xhr.readyState == 4) {
          //5. 处理异步返回的数据
          if(xhr.status == 200) {
            odiv.innerHTML = xhr.responseText;
          }
        }
      }
    </script>
  </body>
</html>
```

```

    }
    else{
        alert("请求失败");
    }
}

//4. 发送请求
xhr.send(null);
</script>
</body>
</html>

```



对于向服务器端的 GET 请求,有时需要在 url 地址后面加上“?”+ 随机数防止被缓存。

7.3.3 提交数据

使用 XMLHttpRequest 对象向服务器发送数据要相对复杂一点,因为有可能要构建请求正文数据。

【例 7-5】 使用 XMLHttpRequest 对象模拟 Form 表单向服务器端提交数据,具体步骤如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>XMLHttpRequest 模拟 Form 表单发送数据</title>
    <style>
      /* CSS 代码略,见配套源码 */
    </style>
  </head>
  <body>
    <form method="post"
action="http://www.meishihui68.com.cn/FormAjax.ashx">
      <fieldset>
        <legend>健康信息</legend>
        身高: <input type="text" name="myheight"
id="myheight"/> cm <br/><br/>
        体重: <input type="text" name="myweight"
id="myweight"/> kg <br/><br/>
        <input type="submit" value="表单提交"/>
        <div class="container">
          <a href="javascript:void(0);" id="btnGet">
XMLHttpRequest GET 提交</a>

```



```

</div>
<div class = "container">
    <a href = "javascript:void(0);" id = "btnPost">
        XMLHttpRequest POST 提交</a>
    </div>
</fieldset>
</form>
<script>
    var heig = document.getElementById("myheight");
    var wei = document.getElementById("myweight");
    var xhr = new XMLHttpRequest();
    document.getElementById('btnGet').addEventListener('click',
    function () {
        PostDataToServer("GET");
    });
    document.getElementById('btnPost').addEventListener('click',
    function () {
        PostDataToServer("POST");
    });
    function PostDataToServer(type){
        var data = heig.name + " = "
            + heig.value + "&" + wei.name + " = " + wei.value;
        var url = "http://www.meishihui68.com.cn/FormAjax.ashx";
        url = type == "GET"?url = url + "?" + data:url;
        xhr.open(type,url,true);
        xhr.onreadystatechange = function(){
            if(xhr.readyState == 4){
                if(xhr.status == 200){
                    alert(xhr.responseText);
                }
            }
        }
        if(type == "GET"){
            xhr.send(null);
        }else if(type == "POST"){
            xhr.setRequestHeader("Content - Type",
                "Application/x-www-form-urlencoded");
            xhr.send(data);
        }
    }
</script>
</body>
</html>

```

页面运行后,显示的效果如图 7-21 所示,可以切换表单的 method 为“get”或“post”,提交数据成功后,页面会跳转,并显示结果;也可以使用 XMLHttpRequest 对象模拟出数据提交(AJAX 程序不会跳转,所以用 alert 弹出了结果):

你的身高是 167cm,体重是 60kg

图 7-21 XMLHttpRequest 模拟 Form 表单提交



代码中有如下这句，这是必须的：

```
xhr.setRequestHeader("Content - Type",
    "Application/x-www-form-urlencoded");
```

使用 Fiddler 抓取 form 表单通信，你会发现 form 表单使用“post”方法提交时，会自动附加请求报头：

```
Content - Type: Application/x-www-form-urlencoded
```

7.3.4 FormData 对象

从例 7-5 可以看出，使用 XMLHttpRequest 对象模拟 Form 表单提交数据给服务器时，无论是“GET”还是“POST”方法，都涉及了数据的拼接问题，因为是字符串，如果数据项较多时，由于数据的键值必须和 Form 表单完全统一，否则服务器端无法处理，所以太容易出错了。HTML5 新标准中提供了一个 FormData 对象，可以用来轻松模拟表单对象中的数据。

1. 创建 FormData 对象

可以使用两种方法创建 FormData 对象。一种是使用 new 关键字创建，页面上不需要 Form 表单，数据需要使用手工方式逐一附加，方法如下：

```
var formData = new FormData();
formData.Append(key, value); //key 是键, value 是对应的数据值
```

另一种方法是直接借助于页面上的 Form 表单，在使用 new 关键字实例化时，将 Form 表单对象直接作为参数，方法如下：

```
var oform = document.getElementById("myForm");
Var formdata = new FormData(oform);
```

2. 发送 FormData 数据

对例 7-5 进行改造，示范如何使用 FormData 数据。

【例 7-6】 向服务器端提交 FormData 数据,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>使用 FormData 传递数据</title>
    <style>
      /* CSS 代码略,见配套源码 */
    </style>
  </head>
  <body>
    <form id="myform">
      <fieldset>
        <legend>健康信息</legend>
        身高:
        <input type="text" name="myheight" id="myheight" /> cm<br/><br/>
        体重:
        <input type="text" name="myweight" id="myweight" /> kg<br /><br />
        <div class="container">
          <a href="javascript:void(0);" id="btnPost">FormData 数据提交</a>
        </div>
      </fieldset>
    </form>
    <script>
      var url = "http://www.meishihui68.com.cn/FormAjax.ashx";
      var xhr = new XMLHttpRequest();
      var oform = document.getElementById("myform");
      document.getElementById('btnPost').addEventListener('click',
      function() {
        xhr.open("POST", url, true);
        xhr.onreadystatechange = function() {
          if(xhr.readyState == 4) {
            if(xhr.status == 200) {
              alert(xhr.responseText);
            }
          }
        }
        var formdata = new FormData(oform);    //使用 FormData 对象
        xhr.send(formdata);
      });
    </script>
  </body>
</html>

```

页面运行后,显示的效果如图 7-22 所示,当数据被 FormData 正确传递后,弹出的结果和例 7-5 是完全一样的。



这个页面的 HTTP 请求在底层处理时和手工发送数据有所不同。

健康信息

身高: cm

体重: kg

FormData数据提交

图 7-22 XMLHttpRequest 发送 FormData 数据

首先,请求报头不需要手工配置,它会自动增加如下报头:

```
Content-Type: multipart/form-data;
boundary = ----WebKitFormBoundaryLhQlPkb1hYSdkmD1
```

发送的请求正文数据是这样包装的:

```
-----WebKitFormBoundaryLhQlPkb1hYSdkmD1
Content-Disposition: form-data; name="myheight"

167
-----WebKitFormBoundaryLhQlPkb1hYSdkmD1
Content-Disposition: form-data; name="myweight"

60
-----WebKitFormBoundaryLhQlPkb1hYSdkmD1 --
```

3. 使用 FormData 对象上传文件

在例 7-6 中,我们看到了一个熟悉的“multipart/form-data”的报头值,这是 form 表单在上传文件时必须使用的 enctype 属性值,也可以使用 XMLHttpRequest 对象借助于 FormData 对象向服务器上传文件。表 7-12 列举了一些与上传数据相关的事件,我们会使用它们完成一个异步上传文件的例子。

表 7-12 XMLHttpRequest 对象与上传文件相关的事件


事件	说 明
progress	进度监测事件,在传送数据的过程中会定期触发,用于返回传送数据的信息。在监测中可以使用该事件的属性计算并显示传送数据的百分比。事件参数 event 中包含了以下几个有用的属性: <ul style="list-style-type: none">lengthComputable: 布尔值,标明是否可以计算传送数据长度,如果是 false,无法计算传送数据百分比;loaded: 已经传送的数据量total: 待传送的数据总量
load	传送数据成功事件
abort	传送数据中断事件
error	传送数据发生错误事件
loadstart	开始传送数据事件

【例 7-7】 使用 FormData 对象实现可以显示进度的文件上传,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>使用 FormData 上传文件</title>
    <style type="text/css">
      /* CSS 代码略,见配套源码 */
    </style>
  </head>
  <body>
    <div class="uploader orange">
      <input type="text" class="filename" readonly id="filename" placeholder="
No file selected..." />
      <input type="button" name="file" class="button" value="浏览" />
      <input type="button" class="button" value="上传"
id="btnUpload" />
      <input type="file" size="30" id="myfile" />
    </div>
    <section class="container">
      <div class="progress">
        <span class="orange" id="pering">
        <span id="pertxt"></span></span>
      </div>
    </section>
    <script>
      var ofile = document.getElementById('myfile');
      var ofileinput = document.getElementById("filename");
      var btnUpload = document.getElementById("btnUpload");
      var pering = document.getElementById("pering");
      var pertxt = document.getElementById("pertxt");
      //选择文件后显示文件路径名
      ofile.addEventListener('change', function() {
        ofileinput.value = this.value;
      });
      //上传按钮事件
      btnUpload.addEventListener("click", function() {
        if(ofileinput.value == "") {
          alert("请选择文件");
          return;
        }
      })
      //过滤为图片
      var mime = ofileinput.value.toLowerCase()
        .substr(ofileinput.value.lastIndexOf("."));
      if(mime == ".jpg" || mime == ".png" || mime == ".gif") {
        var formdata = new FormData();
```

```
formdata.Append("fileToUpload", ofile.files[0]);
var xhr = new XMLHttpRequest();
//为 XMLHttpRequest 对象的 upload 对象配置进度事件监听
xhr.upload.addEventListener("progress",
function(event) {
    if(event.lengthComputable) {
        //处理进度显示
        var per =
            Math.round(event.loaded * 100 / event.total);
        perimg.style.width = per + "%";
        pertxt.innerText = per + "%";
    }
}, false);
//上传完成监听
xhr.addEventListener("load", function() {
    alert("文件上传成功了!");
}, false);
//上传错误监听
xhr.addEventListener("error", function() {
    console.log("上传有错误!");
}, false);
//上传取消监听
xhr.addEventListener("abort", function() {
    console.log("上传取消!");
}, false);
//配置超时处理
xhr.timeout = 5000; //设置超时时间为 5 分
xhr.ontimeout = function() {
    console.log("已超时!");
};
xhr.open("POST",
    "http://www.meishihui68.com.cn/SaveImg.ashx", true);
xhr.send(formdata);
} else {
    alert("只能上传图片");
    return;
}
});
</script>
</body>
</html>
```

页面运行后,可以选择本机的文件向指定的 API 地址进行上传,上传过程中会以进度条动态显示上传进度,如图 7-23 所示。

 用 Fiddler 抓取这个例子的 HTTP 请求,会发现 XMLHttpRequest 实际发出了

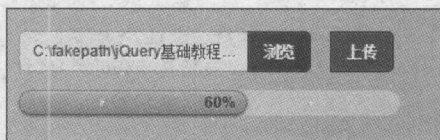


图 7-23 XMLHttpRequest 上传文件

两次请求，第一次使用的是“OPTIONS”请求方法，用于探测 API 是否可用，第二次才是“POST”方法，用于上传数据。所以服务器端 API 一定要配置允许“OPTIONS”请求。

7.3.5 解析 XML 数据

当前的移动应用开发更倾向于使用 JSON 格式的数据，AJAX 的缩写即为 Asynchronous JavaScript and XML，过去的 API，特别是 Web Services 大都返回 XML 数据。XML 数据也有一些明显的优势，例如信息检索较方便、可扩展性较强等，如图 7-24 所示，在 taobao 提供的 API 中，数据返回格式是可以根据需要设定为 XML 或 JSON 格式的。

返回格式(Format): XML

API 类目(API Category): JSON

API 名称(API Name): taobao.wlb.order.consign

数据环境(environment): ☐ 沙箱(Sandbox) ☒ 正式(Online)

提交方式(Method): ☒ POST ☐ GET

SDK 类型(SDK Language): ☒ JAVA ☐ PHP ☐ .NET ☐ PYTHON

AppKey: 系统分配(Default) 自定义(Custom Settings)

AppSecret: 系统分配(Default)

将鼠标移至说明上，查看参数介绍；* 表示必填，* 表示几个参数中必填一个；查看 API 详情
Mouse over the caption line to know Parameter Reference, View API Detail

wlb_order_code: *

验证码: 1111 看不清楚?

提交测试(Execute) 绑定用户(Authorize)

图 7-24 taobao API 数据返回格式设定

当使用 XMLHttpRequest 请求数据，返回的是 XML 格式时，就需要用到它的另一个属性 responseXML，这个属性代表一个 XML DOM 对象，可以专门用来解析 XML 格式数据，这个 XML DOM 对象提供了一个 getElementsByTagName 方法（和 HTML DOM 的方法一样），可以根据标签名来得到数据（还可以使用 XPath 查询，限于篇幅，本书不作介绍）。下面来看一个解析 XML 格式数据的例子。

【例 7-8】 使用 XMLHttpRequest 对象实现请求并解析 XML 格式数据，先来看需要请求得到的 XML 数据，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<bookstore>
  <book category="烹饪">
    <title>家常菜精选 1288 例</title>
    <price>29.80</price>
  </book>
  <book category="魔幻">
    <title>哈利波特</title>
```



```

    <price>22</price>
  </book>
  <book category="Web 开发">
    <title>JavaScript 高级程序设计</title>
    <price>59.00</price>
  </book>
  <book category="Web 开发">
    <title>AJAX 程序设计</title>
    <price>49</price>
  </book>
</bookstore>

```

从这个 XML 数据中可以看到 4 本书的信息,下面使用 XMLHttpRequest 请求远程数据,并使用表格进行展示,页面代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>XMLHttpRequest 对象处理 xml 数据</title>
    <style>
      /* CSS 代码略,见配套源码 */
    </style>
  </head>
  <body>
    <table id="dataTable">
      <thead>
        <th class="bookname">书籍名称</th>
        <th class="bookprice">价格</th>
      </thead>
      <tbody>
      </tbody>
    </table>
    <script>
      var tbody=document.querySelector("tbody");
      var xhr=new XMLHttpRequest();
      xhr.open("GET",
http://www.meishihui68.com.cn/books.xml?
+ Math.random(),true);
      xhr.onreadystatechange=function(){
        if(xhr.readyState==4){
          if(xhr.status==200){
            //获得 xmldom 对象
            var oxmldom=xhr.responseXML;
            var books=oxmldom.getElementsByTagName("book");
            for(var i=0;i<books.length;i++){

```

```

        var book = books[i];
        var bookname =
book.getElementsByTagName("title")[0].childNodes[0].nodeValue;
        var bookprice =
book.getElementsByTagName("price")[0].childNodes[0].nodeValue;
        var otr = document.createElement("tr");
        var otd1 = document.createElement("td");
        otd1.innerText = bookname;
        otr.appendChild(otd1);
        var otd2 = document.createElement("td");
        otd2.innerText = "¥" + bookprice;
        otr.appendChild(otd2);
        tbody.appendChild(otr);
    }
}
xhr.send(null);
</script>
</body>
</html>

```

页面运行效果如图 7-25 所示。

书籍名称	价格
家常菜精选1288例	¥29.80
哈利波特	¥22
JavaScript高级程序设计	¥59.00
AJAX程序设计	¥49

图 7-25 XMLHttpRequest 请求并解析 XML 数据



要想使用 responseXML 处理 xml 格式的数据，服务器端返回数据时，响应报头必须这样设置：

```
Content-Type: text/xml 或者 Content-Type: Application/xml
```

7.4 CORS 跨域问题

使用 AJAX 进行数据通信时，不可避免地会遇到跨域(Cross-origin resource sharing, CORS)问题。什么是跨域问题呢？将例 7-4 中请求地址直接修改成 <http://www.baidu.com>，会发现程序不再运行，打开 Chrome 的控制台，会发现报错信息：

```
XMLHttpRequest cannot load https://www.baidu.com/. No
'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'null' is
therefore not allowed access.
```

从报错信息中大概可以看出对于请求的网址，服务器的响应少了报头。本书提供的所有外网 API 都作了允许跨域的处理，使用 Fiddler 抓取后，会发现每个 API 加了以下 3 个特殊的响应报头：

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Content-Type
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
```

- Access-Control-Allow-Origin：用来控制请求来自哪个源(源的信息包括协议、域名和端口)，根据这个值，服务器决定是否同意这次请求。* 表示所有，也可以单独设定某些网站，在许可范围内，服务器会返回一个正常的 HTTP 回应。如果响应的报头信息没有包含 Access-Control-Allow-Origin 报头，就知道不能跨域，从而抛出一个错误。
- Access-Control-Allow-Headers：可选报头。CORS 请求时，XMLHttpRequest 对象的 getResponseHeader() 方法只能得到 6 个基本字段：Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma。如果想得到其他字段，就必须在 Access-Control-Expose-Headers 里面指定。
- Access-Control-Allow-Methods：逗号分隔的一个字符串，表明服务器支持的所有跨域请求的方法。

对于跨域的判断，Web 浏览器对于相同的域只是通过 URL 的首部来识别，不会去尝试判断相同的 IP 地址是否对应两个域。以下是一些跨域的实际例子，见表 7-13，假设在源上负责生成 XMLHttpRequest 对象，向目标地址发出请求。

表 7-13 跨域的一些实例

请求的源及目标	结 论
源：http://www.mysite.com/1.html 目标：http://www.mysite.com/2.html	同域名同端口，同域
源：http://www.mysite.com:8080/1.html 目标：http://www.mysite.com/2.html	端口不一致，跨域
源：http://www.mysite.com/1.html 目标：https://www.mysite.com/2.html	协议不匹配，跨域
源：http://www.mysite.com/1.html 目标：http://192.168.0.1/2.html	跨域，即使 www.mysite.com 的 IP 地址是 192.168.0.1，但浏览器不会知道
源：http://www.mysite.com/1.html 目标：http://scripts.mysite.com/2.html	跨域，子域被视为其他域
源：http://www.mysite.com/1.htm 目标：http://www.myasp.com/2.htm	跨域，即使 www.mysite.com 和 www.myasp.com 是同一个网站，但浏览器不会知道

7.5 RESTful API 介绍

移动应用程序分为前端和后端两个部分。当前的发展趋势,就是前端设备层出不穷,例如手机、平板、台式计算机、其他专用设备……因此,必须有一种统一的机制,方便不同的前端设备与后端进行通信。这导致 API 架构的流行,甚至出现“API First”的设计思想。RESTful 架构就是目前最流行的一种互联网软件架构。它结构清晰,符合标准,易于理解,扩展方便,所以正得到越来越多网站的采用。

RESTful 可以认为是一种建立在 HTTP 协议之上的设计模式,充分地利用了 HTTP 协议的特点,使用 URL 来表示资源,用各个不同的 HTTP 动词来表示对资源的各种行为。这样做的好处就是资源和操作分离,让资源的管理更加规范。REST 的核心是资源,并且资源是用统一资源定位符 URLs 来标识的。从概念上来讲,资源和它的状态(提供给客户的格式)是分开的。REST 不做任何格式上的要求,但是一般包含 XML 和 JSON。

RESTful API 是目前比较成熟的一套互联网应用程序的 API 设计理念,API 与用户的通信协议总是使用 HTTP 协议。对于资源的具体操作类型,由 HTTP 动词表示,通过 GET、DELETE、POST、PUT 来请求 URL 资源,由它来对应业务对象的查询、删除、生成、修改,数据也由 URL 地址或请求正文进行传递。



RESTful API 返回的数据格式是 XML 或 JSON,取决于 HTTP 请求消息的报头 Content-Type 的设置是 Application/xml 或 Application/json。

7.6 jQuery 中的 AJAX 方法

jQuery 确实是一个优秀的轻量级的 JS 框架,能帮助我们快速地开发 JS 应用,并在一定程度上改变了人们写 JavaScript 代码的习惯,对于 AJAX 通信,jQuery 也作了完美的封装,主要提供了以下几种方法。

(1) \$.get(url,[data],[callback],[type]): 通过远程 HTTP GET 请求信息。

url: 发送请求的 URL 地址。

data: (可选)要发送给服务器的数据,以 key/value 的键值对形式表示,数据会自动转化成如 key1=value1&key2=value2&...的形式,数据会作为 QueryString 附加到请求 URL 中。

callback: 请求成功时的回调函数,响应数据会作为第一个参数进行传递,如果指定了 type 属性,data 会自动转换成相应对象。

type: 返回内容格式,例如 xml, html, script, json, text 等。

下面是一段请求 API 的示范代码:

```
$.get("http://www.meishihui68.com.cn",
    {name:"huangbo",age:40},
    function (data){
        //处理 data
```

```
    }  
};
```

(2) `$.post(url,[data],[callback],[type])`: 通过远程 HTTP POST 请求信息。

url: 发送请求的 URL 地址。

data: (可选)要发送给服务器的数据,以 key/value 的键值对形式表示,数据会自动转变成形如 `key1=value1&key2=value2&...` 的形式,以请求正文形式附加到请求中。

callback: 请求成功时的回调函数,响应数据会作为第一个参数进行传递,如果指定了 type 属性,data 会自动转换成相应对象。

type: 返回内容格式,例如 xml、html、script、json、text 等。

下面是一段请求 API 的示范代码:

```
$.post("http://www.meishihui68.com.cn",  
      {name:"huangbo",age:40},  
      function (data){  
          //处理 data  
      }  
);
```

(3) `$.getJSONurl,[data],[callback]`: 通过远程 HTTP GET 请求 JSON 格式数据。

url: 发送请求的 URL 地址。

data: (可选)要发送给服务器的数据,以 key/value 的键值对形式表示,数据会自动转变成形如 `key1=value1&key2=value2&...` 的形式,数据会作为 QueryString 附加到请求 URL 中。

callback: 请求成功时的回调函数,响应数据会作为第一个参数进行传递。

下面是一段请求 API 的示范代码:

```
$.getJSON("http://www.meishihui68.com.cn",  
          {name:"huangbo",age:40},  
          function (data){  
              //处理传入的 JSON 对象 data  
          }  
);
```

(4) `$.ajax(settings)`: 这是 jQuery 中实现 AJAX 通信的底层实现。它可以实现更丰富的配置,以获得最大的灵活性。

settings: AJAX 请求设置,是个 JSON 对象,它有以下设置。

- url: 请求的地址。
- method: 所用的 HTTP 请求方法。
- cache: 布尔值,用于指明是否缓存请求,主要针对 HTTP GET 请求,在 URL 地址后面加上随机数。
- contentType: 用于设定 ContentType 请求报头的值。

- **dataType**: 预期返回的数据类型,请求成功的回调函数会为依据这个配置对返回的数据进行处理,“xml”会传入处理后的 XML DOM 对象,“json”会传入处理后的 json 对象,“html”传入纯文本的 HTML 信息,包含的< script >会在 DOM 插入时运行,“script”传入纯文本的 JavaScript 代码,“text”返回纯文本。
- **data**: HTTP 请求中要传递的数据,可以是字符串,也可以是 json 对象,后者会自动转成 key1=value1&key2=value2&...的形式。
- **timeout**: 设置请求超时时间(毫秒)。
- **headers**: 请求时需要额外添加的请求报头,以 Json 对象方式配置。
- **success**: 请求成功时的回调函数配置,可以使用匿名函数或函数名,函数的第 1 个参数是返回的响应正文或根据 dataType 处理好的对象。
- **error**: 请求失败时的回调函数配置,可以使用匿名函数或函数名,有 3 个输入参数: XMLHttpRequest 对象、错误信息、(可选)捕获的异常对象。如果发生了错误,错误信息(第 2 个参数)除了得到 null 之外,还可能是“timeout”“error”“notmodified”和“parsererror”。

下面是一段请求 API 的示范代码:

```
$.ajax({
  url:"http://www.meishihui68.com.cn",
  method:"GET",
  dataType:"json",
  data: {name:"huangbo",age:40},
  timeout:2000,
  success:function(res){
    //处理 JSON 对象 res
  },
  error:function(xhr,e){
    //处理错误信息
  }
});
```

7.7 实例：送货地址管理

表 7-14 列出了一个典型的 RESTFul API 的示例,将使用 jQuery 的 AJAX 通信的各种方法,按说明向 API 发出不同的 HTTP 请求以实现送货地址的增、删、查、改。

表 7-14 RESTFul API 示例

请求方法	资源 URL	说 明
GET	http://http://www. meishihui68. com. cn/api/RESTFul	得到所有收货地址
POST	http://http://www. meishihui68. com. cn/api/RESTFul	新增一个收货地址,请求正文是个 json 对象,格式为 {"No":id 编号," Receiver": 收货人," Tel": 电话, "Address":地址}

续表

请求方法	资源 URL	说 明
PUT	http://http://www. meishihui68. com. cn/api/RESTFul/{id}	修改一个编号为 id 的收货地址, 请求正文是个 json 对象, 格式为{"No":id 编号,"Receiver":收货人,"Tel":电话,"Address":地址}
DELETE	http://http://www. meishihui68. com. cn/api/RESTFul/{id}	删除编号为 id 的收货地址, id 是整数

【例 7-9】 jQuery 与 RESTFul API 的通信, 实现送货地址管理, 由于 CSS 和 HTML 代码都比较多, 就不在此赘述, 请参考本书的配套源代码。

页面运行后, 如图 7-26 所示, 页面会自动向 API 请求数据生成送货地址列表, 也可以输入新的送货人、电话和收货地址进行添加, 还可以修改或删除某个收货地址。使用 jQuery 封装的各种 AJAX 通信方法, 能很轻松地完成与 RESTFul API 进行各种 HTTP 请求, 从而完成业务对象的各种处理。

收货地址管理

收货人: 收件人姓名

电话: 收件人电话

收货地址: 收件人送货地址

增加









收货人	电话	收货地址	操作
黄波	13611111111	四川省成都市天府广场1号	 
张小华	13622222222	四川省成都市天府广场2号	 
黄平	13633333333	四川省成都市天府广场3号	 
张铁	13644444444	四川省成都市天府广场4号	 

图 7-26 送货地址管理

小结

本章主要讲解了 AJAX 通信技术。先简单介绍了 AJAX 技术, 并借助强大的工具 Fiddler 熟悉了通信的底层——HTTP 协议, 讲解了 AJAX 通信中的核心对象 XMLHttpRequest 对象, 以及如何使用它来读取和发送数据, 并讲解了 FormData 的使用、

XML 数据的处理,以及在通信中遇到的 CROS 跨域问题,最后简单介绍了 RESTful API 的概念和 jQuery 中的各种 AJAX 通信方法,并给出一个综合的案例:实现送货地址的管理。

习题

一、选择题

1. 以下()技术不是 AJAX 通信技术的组成部分。
A. XMLHttpRequest B. DHTML C. CSS D. JavaScript
2. XMLHttpRequest 对象有()个状态返回值。
A. 3 B. 4 C. 5 D. 6
3. 下面代码中,存在错误的选项是()。

```
var url = "?operate=doCheckUserExists&uname=" + uname;
var xmlhttpRequest = new XMLHttpRequest();
xmlHttpRequest.onreadystatechange = dealRes;
xmlHttpRequest.open("POST",url,false);
xmlHttpRequest.send(url);
```

- A. onreadystatechange 应为 onReadyStateChange
 - B. 发送请求的方式应为 GET
 - C. open 方法的第 3 个参数应该是 true,因为要异步发送请求
 - D. send 方法应该传入 null 参数,而不是将 url 当作参数
4. 在 AJAX 模式中,客户端的请求是()完成的。
A. 同步 B. 并发 C. 异步 D. 单向
 5. 在创建请求代码片段,如下:

```
xhr.open("GET","http://www.meishihui68.com.cn/wa.aspx?b=1",true),
```

- 传递的参数值为()。
- A. get
 - B. b
 - C. wa.aspx
 - D. 1
6. 在 AJAX 技术中,关于 HTTP 协议向服务器传送数据的方式,描述正确的是()。
A. 包括 POST、GET 两种方式
B. 如果传输数据包含机密信息,建议采用 MD5 数据提交方式
C. GET 执行效率和 POST 方法一样
D. POST 方式传送的数据量较小,不能大于 1B
 7. 使用 XMLHttpRequest 发送请求不包括()选项
A. 直接扩展 jQuery 函数 B. 创建 XMLHttpRequest 对象
C. 设置回调函数 D. 使用 send()方法发送请求
 8. 在 AJAX 技术中,获取服务器端回传的响应正文,应该采用 XMLHttpRequest 对象

的()。

- A. responseXML 属性
- B..responseText 属性
- C. responseValue 属性
- D. getXml

9. 在 AJAX 中可以使用 FormData 传递数据,下面()选项是正确的。

- A. 可以使用 FormData 上传文件
- B. 使用 FormData 处理数据,类似 form 表单中 enctype="multipart/form-data"
- C. FormData 数据的添加既可以直接使用 form,也可以使用 Append 方法
- D. 以上说法全部正确

10. 在 jQuery 的 ajax 方法中,使用()属性对要发送的数据进行配置。

- A. dataType
- B. data
- C. formdata
- D. success

二、判断题

1. AJAX 程序可以向任何 API 或网址无障碍进行通信。()
2. AJAX 是全新的技术,和以前的 JavaScript 没什么关系。()
3. 使用 AJAX 把数据发送给服务器,数据必须使用请求正文。()
4. XMLHttpRequest 对象只能实现异步请求。()
5. RESTful API 返回的数据格式既可以是 XML,也可以是 JSON。()

三、填空题

1. AJAX 的英文全称是_____。
2. 要使用 XMLHttpRequest 对象的 responseXML 属性处理 XML 格式的数据,响应正文的 Content-Type 必须为_____或_____。
3. HTTP 协议的响应状态表示资源没找到的代码是_____。
4. HTTP 协议是由_____和_____两部分构成的,是_____的协议。
5. jQuery 在 AJAX 通信中,如果希望在回调函数中能直接处理 JSON 对象,需要把_____属性设置为_____。
6. 在 RESTful API 调用中,一般通过 HTTP 请求_____方法实现删除。
7. 如果在浏览器中希望 AJAX 实现 CROS(跨域)通信,服务器端必须进行配置,保证响应消息中有报头_____、_____和_____。

四、简答题

1. 简述 AJAX 的工作原理。
2. 简述 XMLHttpRequest.status 常见的属性值有哪些。
3. jQuery 中实现 AJAX 通信的方法有哪些? 如何使用?
4. 什么是 RESTful API? 这种 API 有哪些特点?

五、编程题

使用 GET 方法访问下列 API 地址:

```
http://www.meishihui68.com.cn/api/tel?tel =
```

使用 jQuery 中的 AJAX 通信方法,实现如图 7-27 所示的效果。

手机归属地查询	
电话：13021671512	
	查询
号码归属地：	山东 青岛市
提供商：	联通如意通卡

图 7-27 手机归属地查询

第 8 章

CHAPTER 8

WebSocket 基础

学习目标

- 了解 WebSocket 的由来。
- 了解 WebSocket 技术的基本原理。
- 熟练掌握 WebSocket 的 API。

作为新一代的 Web 标准,HTML5 拥有许多引人注目的新特性,其中有“Web TCP”之称的 WebSocket 格外吸引开发人员的注意。本章将介绍 HTML5 WebSocket 的基本原理和编程接口,并以一个简单的“聊天室”案例来示范怎样实现一个 WebSocket 应用。

8.1 WebSocket 的发展历程

众所周知,Web 应用的交互过程通常是客户端通过浏览器发出一个请求,服务器端接收请求后进行处理并返回结果给客户端,客户端浏览器将信息呈现,这种机制对于信息变化不是特别频繁的应用尚可,但对于实时要求高、海量并发的应用来说显得捉襟见肘,尤其在当前业界移动互联网蓬勃发展的趋势下,高并发与用户实时响应是 Web 应用经常面临的问题,例如金融证券的实时信息,Web 导航应用中的地理位置获取,社交网络的实时消息推送等。

传统的请求/响应的 HTTP 协议模式的 Web 开发在处理此类业务场景时,通常采用的技术方案有两种。

(1) 轮询。原理简单易懂,就是客户端通过一定的时间间隔以频繁请求的方式向服务器发送请求,来保持客户端和服务器端的数据同步。问题很明显,当客户端以固定频率向服务器端发送请求时,服务器端的数据可能并没有更新,带来很多无谓请求,浪费带宽,效率低下。

(2) 基于 Flash。Adobe Flash 通过自己的 Socket 实现完成数据交换,再利用 Flash 暴露出相应的接口为 JavaScript 调用,从而达到实时传输目的。此方式比轮询要高效,且因为 Flash 安装率高,应用场景比较广泛,但移动互联网终端对于 Flash 的支持并不好。iOS 系统中没有 Flash 的存在,在 Android 中虽然有 Flash 的支持,但实际的使用效果差强人意,且对移动设备的硬件配置要求较高。2012 年 Adobe 官方宣布不再支持 Android 4.1+ 系统,宣告了 Flash 在移动终端上的死亡。

传统 Web 模式在处理高并发及实时性需求的时候,会遇到难以逾越的瓶颈,需要一种

高效节能的双向通信机制来保证数据的实时传输。在此背景下,基于 HTML5 规范的、有“Web TCP”之称的 WebSocket 应运而生。

HTML5 WebSocket 的设计目的就是要取代轮询和 Flash 技术,使客户端浏览器具备像 C/S 架构下桌面系统的实时通信能力。浏览器通过 JavaScript 向服务器发出建立 WebSocket 连接的请求,连接建立以后,客户端和服务端就可以通过 TCP 连接直接交换数据。因为 WebSocket 连接的本质就是一个 TCP 连接,所以在数据传输的稳定性和数据传输量的大小方面,和轮询以及 Comet 技术比较,它具有很大的性能优势。

8.2 HTML5 WebSocket 简介

WebSocket 是 HTML5 一种新的协议,它实现了浏览器与服务器全双工通信,能更好地节省服务器资源和带宽并实现实时通信,它建立在 TCP 之上,同 HTTP 一样通过 TCP 来传输数据,但是它和 HTTP 协议最大的不同在于:

- (1) WebSocket 是一种双向通信协议,在建立连接后,WebSocket 服务器和浏览器或其他客户端都能主动地向对方发送或接收数据,就像 Socket 一样;
- (2) WebSocket 需要类似 TCP 的客户端和服务端通过握手连接,连接成功后才能相互通信。

相对于传统 HTTP 每次请求和响应都需要客户端与服务器端建立连接的模式,WebSocket 是类似 Socket 的 TCP 长连接的通信模式,一旦 WebSocket 连接建立后,后续数据都以帧序列的形式传输。在客户端断开 WebSocket 连接或服务器端断掉连接之前,不需要客户端和服务端重新发起连接请求。在海量并发及客户端与服务器交互负载流量大的情况下,极大地节省了网络带宽资源的消耗,有明显的性能优势,且客户端发送和接受消息是在同一个持久连接上发起,实时性优势明显。如图 8-1 所示,这两者的交互有很大不同。

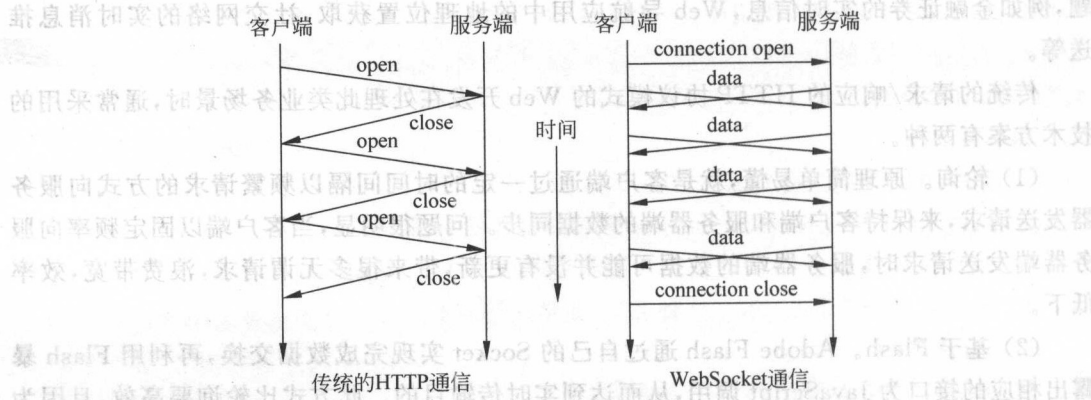


图 8-1 传统的 HTTP 通信和 WebSocket 通信对比

WebSocket 协议可以在现有的 Web 基础结构下很好地工作,它定义 WebSocket 连接的生命周期始于 HTTP 连接,从而保证了与之前的 Web 应用程序的兼容性。在经过 WebSocket 握手后,协议才从 HTTP 切换到 WebSocket。下面使用 Fiddler 抓取客户端和

服务器端交互的报文查看 WebSocket 通信与传统 HTTP 通信的不同:

```
GET http://127.0.0.1:7999/ HTTP/1.1
Host: 127.0.0.1:7999
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
Origin: http://localhost:63342
Sec-WebSocket-Version: 13
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.143 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Sec-WebSocket-Key: p947dnotLEAaxiOEecBTBw==
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits
```

可以看到,客户端发起的 WebSocket 连接报文类似传统 HTTP 报文,其中的请求报头“Upgrade: websocket”表明这是 WebSocket 类型请求,“Sec-WebSocket-Key”是客户端发送的一个 base64 编码的密文,要求服务器端必须返回一个对应加密的响应报头“Sec-WebSocket-Accept”应答,否则客户端会抛出“Error during WebSocket handshake”错误,并关闭连接。

服务器端收到报文后返回的响应消息格式类似:

```
HTTP/1.1 101 Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept: JIK21sSenzmiRDegToT4ZpDtrC8 =
```

此时,HTTP 连接将终止,并被 WebSocket 连接替代。WebSocket 连接与之前的 HTTP 连接都基于相同的 TCP 连接。一旦建立好连接,WebSocket 数据帧就可以在客户端和服务器之间以全双工模式发送和传回。文本和二进制数据帧可以同时传输。数据帧最少包含 2 字节数据。如果是文本数据帧,则以 0x00 开头,以 0xFF 结束,中间包含 utf-8 数据。

8.3 WebSocket 实现

WebSocket 的实现分为客户端和服务端两部分,客户端(通常为浏览器)发出 WebSocket 连接请求,服务器端响应,实现类似 TCP 握手的动作,从而在浏览器客户端和 WebSocket 服务器端之间形成一条 HTTP 长连接快速通道。两者之间后续将直接进行数据互相传送,不再需要发起连接和相应。

WebSocket 服务器端在各个主流应用服务器厂商中已基本获得支持,以下列举了部分

常见的商用及开源应用服务器对 WebSocket 服务器端的支持情况,如表 8-1 所示。

表 8-1 WebSocket 服务器端支持情况

应用服务器	支持情况
WebSphere	WebSphere 8.0 以上版本支持,7.X 之前版本结合 MQTT 支持类似的 HTTP 长连接
WebLogic	WebLogic 12c 支持,11g 及 10g 版本通过 HTTP Publish 支持类似的 HTTP 长连接
IIS	IIS 7.0+ 支持
Tomcat	Tomcat 7.0.5+ 支持,7.0.2X 及 7.0.3X 通过自定义 API 支持
Jetty	Jetty 7.0+ 支持
Node.js	需要加载外置的 WebSocket 库支持

对于 WebSocket 客户端,主流的浏览器(包括 PC 和移动终端)现已都支持标准的 HTML5 的 WebSocket API,这意味着客户端的 WebSocket JavaScript 脚本具备良好的一致性和跨平台特性,以下列举了常见的浏览器厂商对 WebSocket 的支持情况,如表 8-2 所示。

表 8-2 主流的浏览器对 WebSocket 支持情况

浏览器	WebSocket 支持情况
Chrome	4.0 及以上版本
FireFox	5.0 及以上版本
Internet Explore	10.0 及以上版本
Opera	10.0 及以上版本
Safari	iOS 5.0 及以上版本
Android Browser	Android 4.5 及以上版本

以下以一段代码示例说明 WebSocket 的客户端实现:

```
var ws = new WebSocket("ws://echo.websocket.org");
ws.onopen = function(){
    ws.send("test!");
};
ws.onmessage = function(msg){
    console.log(msg.data);
};
ws.onclose = function(){
    console.log("WebSocketClosed!");
};
ws.onerror = function(){
    console.log("WebSocketError!");
};
```

第一行代码是在生成一个 WebSocket 对象,参数是需要连接的服务器端的地址,同 HTTP 协议开头一样,WebSocket 协议的 URL 使用 ws:// 开头,另外安全的 WebSocket 协议使用 wss:// 开头。

第二行到最后一行为 WebSocket 对象注册消息的处理函数,WebSocket 对象一共支持 4 个消息: `onopen`、`onmessage`、`onclose` 和 `onerror`。有了这 4 个事件,就可以很容易很轻松地驾驭 WebSocket。当浏览器和 WebSocket 服务器连接成功后,会触发 `onopen` 消息;如果连接失败,发送、接收数据失败或者处理数据出现错误,会触发 `onerror` 消息;当浏览器接收到服务器发送过来的数据时,就会触发 `onmessage` 消息,参数 `msg` 中的 `data` 属性包含了服务器传输过来的数据;当浏览器接收到服务器发送的关闭连接请求时,就会触发 `onerror` 和 `onclose` 消息。可以看出,所有的事件都采用异步回调的方式触发,这样不会阻塞 UI,可以获得更快的响应时间以及更好的用户体验。

8.4 实例：聊天室

在本节将以一个“聊天室”为例,说明 HTML5 标准中的 WebSocket 的优势及具体开发实现。聊天室中的用户发言后,其他用户的客户端(Android、iOS、PC 浏览器或手机浏览器端)都会迅速显示聊天内容,新用户进入或用户关闭程序或浏览器,聊天室都能即时显示通知消息。

8.4.1 WebSocket 服务器端

WebSocket 通信是客户端和服务器双方的事情,仅编写客户端代码是不够的,还需要准备好相应的 WebSocket 服务器,才能实现客户端和服务器之间的 WebSocket 通信。WebSocket 是 HTML5 标准中的一个协议,从理论上来说,用哪种技术实现服务器端并不重要。本节使用 Node.js 来搭建 WebSocket 服务器。

这里简单介绍 Node.js。JavaScript 高涨的人气带来了很多变化,以至于如今使用其进行网络开发的形式也变得截然不同了。过去从前端跨越到后端,前端使用 JavaScript 编程,而后端必须使用其他技术(例如 C#、Java、PHP 等),这样巨大的反差让人难以想象和跨越,开发人员不得不掌握多门语言。Node.js 是一个 JavaScript 运行环境(Runtime),它实际上是对 Google V8 引擎进行了封装。V8 引擎执行 JavaScript 的速度非常快,性能非常好,而 Node.js 对一些特殊用例进行了优化,提供了替代的 API,使得 V8 在非浏览器环境下运行得更好。Node.js 可用于方便地搭建响应速度快、易于扩展的网络应用,它使用事件驱动,非阻塞 I/O 模型而得以轻量 and 高效,非常适合在分布式设备上运行数据密集型的实时应用。这意味着,基于 Node.js,也可以让 JavaScript 运行在服务器上,充当高效的服务器端。

有兴趣的读者可以研究 Node.js(<https://nodejs.org/en/>)技术。由于篇幅有限,这里就不具体介绍 Node.js 的其他内容了。

“聊天室”的服务器端配置如下:

- (1) 从 Node.js 官网下载 Node.js 的安装程序,将其安装在 Windows 操作系统中。
- (2) 将本书的配套资源包中提供的服务器端“ImChatServer.rar”解压到一个目录,例如在目录“G:\Mybook\ImChatServer”中,文件内容如图 8-2 所示。
- (3) 在服务器上安装完后,在开始菜单中,找到 Node.js 的安装项,选择“Node.js command prompt”,如图 8-3 所示。

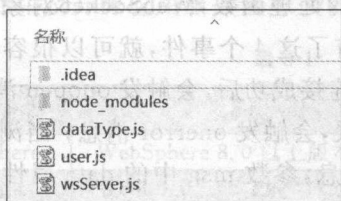


图 8-2 聊天室服务器端程序

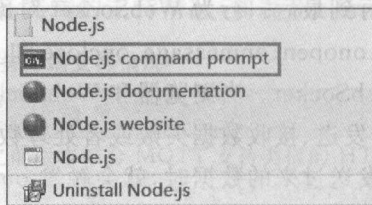


图 8-3 Node.js 安装项

(4) 在弹出的 Node.js 命令行窗口中输入命令“node -v”，如果显示 Node.js 的版本号，则 Node.js 安装成功，如图 8-4 所示。

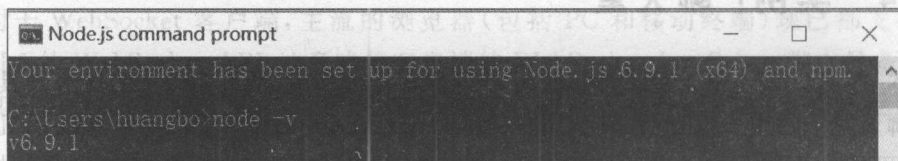


图 8-4 Node.js 版本显示

(5) 再次进入 Node.js 的命令行窗口，进入服务器端所在的目录后，输入命令“node wsServer.js”，回车后运行，如果显示“Websocket Server is running on 1234”，则说明服务器端运行成功，侦听端口为“1234”，如图 8-5 所示。

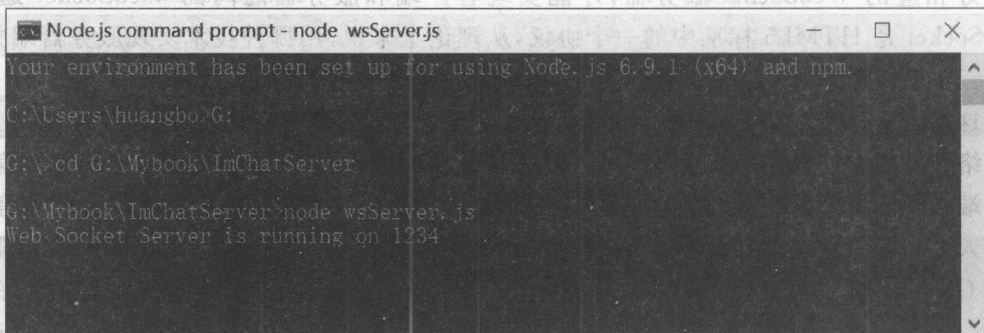


图 8-5 服务器端运行效果

(6) 记录下服务器的 IP 地址，例如：192.168.1.7。

8.4.2 客户端实现

1. 数据类型以及消息格式的定义

在这个 App 应用中消息传递有好几种：用户信息、聊天信息、确认信息、欢迎信息，为了能让服务器和客户端能正确区分各种消息，必须定义一个消息枚举，以及对消息的格式实现预定义，“dataTypes.js”的文件内容如下：

```
//定义消息类型枚举
var messageType = {
```

```
userInfo:0,      //用户信息
chatInfo:1,      //聊天信息
confirmInfo:2,   //确认信息
welcomeInfo:3    //欢迎信息
}
//用户信息数据格式
var userInfo = {
  msgType:messageType.userInfo,
  userName:null,
  userPhoto:null
};
//聊天信息数据格式
var chatInfo = {
  msgType:messageType.chatInfo,
  chatMessage:null,
  userName:null,
  userPhoto:null,
  chatDateTime:null
};
//确认消息数据格式
var confirmInfo = {
  msgType:messageType.confirmInfo,
  confirm:false,
  userToken:null
};
//欢迎消息数据格式
var welcomeInfo = {
  msgType:messageType.welcomeInfo,
  content:null
};
```

2. 页面设计以及核心代码

在“app.js”中对用户与浏览器之间的数据传递采用了 WebSocket 的 API 进行了通信(WebSocket 服务器端地址要修改成实际的服务器地址和端口)。当消息发送给 Node.js 充当 WebSocket 服务器后,由服务器端 JavaScript 判断处理,并向其他客户端进行了消息传递。由于页面 HTML 以及 JS 代码较多,就不在此赘述了,请参考本书的配套源代码。

3. 实现效果

配置并运行了 Node.js 服务器端后,启动客户端,自动和 WebSocket 服务器端连接,如果连接成功,系统会显示服务器端模拟登录后分配的用户名和头像,单击“点击进入聊天室”按钮,进入聊天室,这时其他已连接的客户端会自动在顶部导航栏中显示新用户进入聊天室的欢迎信息;如果连接不成功或在聊天中连接断开,系统自动提示“服务器无法连接!”。在聊天室中用户的聊天信息都会及时显示在客户端界面上,效果如图 8-6 所示。

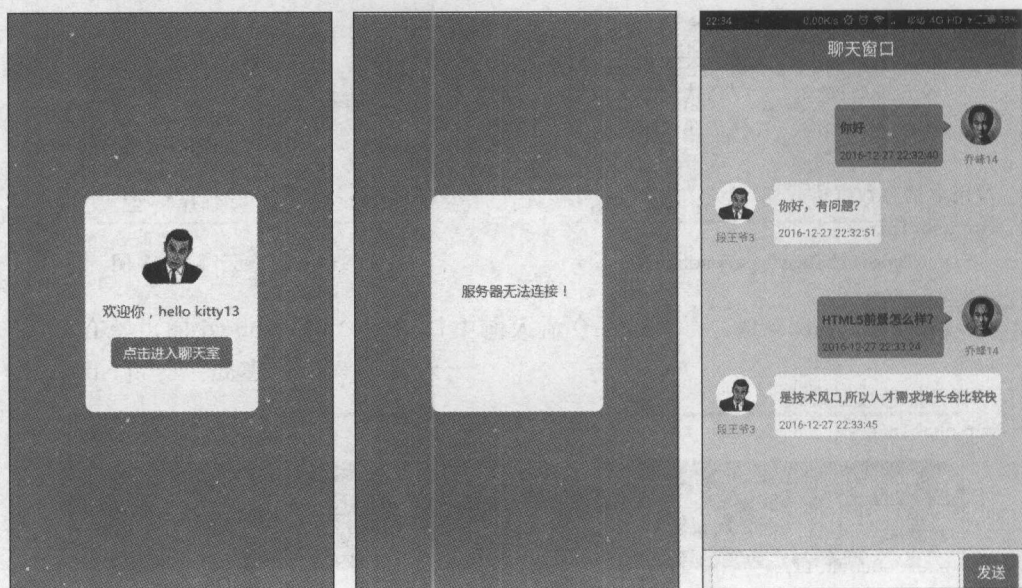


图 8-6 聊天室运行效果

小结

本章介绍了 HTML5 WebSocket 的发展历程以及它尝试解决的问题,然后介绍了 HTML5 WebSocket 标准和 WebSocket 编程接口,并且演示了怎样使用 WebSocket 构建一个实时的“聊天室”应用。当前已经可以在各种移动设备、平板计算机上看到 HTML5 WebSocket 的身影,WebSocket 将会成为未来开发实时 Web 应用的生力军,作为 HTML5 开发人员,关注其中的 WebSocket 标准也应该提上日程了,否则将在新一轮的软件革新的浪潮中被淘汰。

习题

一、选择题

- WebSocket 连接服务器时的正确语法形式应该为()。
 - `var ws=new WebSocket("ws://127.0.0.1:3999");`
 - `var ws=new WebSocket("http://127.0.0.1:3999");`
 - `var ws=new webSocket("http://127.0.0.1:3999");`
 - `var ws=new webSocket("ws://127.0.0.1:3999");`
- 通过 WebSocket 技术可以用来实现以下()类型的项目。
 - 游戏
 - 聊天室
 - 股票行情
 - 以上都可以
- WebSocket API 中使用()事件对服务器传递过来的消息进行处理。
 - onopen
 - onclose
 - onmessage
 - onerror

二、判断题

1. JavaScript 只能运行在浏览器上。(使用) 一个能识别的格式。例如下面这段代码:
2. HTML5 的 WebSocket 服务器端只能使用 Node.js 实现。()
3. WebSocket 通信和 HTTP 通信一点关系都没有。()
4. WebSocket 通信中只能由服务器推送消息给客户端。()
5. 目前只有 Chrome 浏览器支持 HTML5 WebSocket。()

三、填空题

1. 过去 HTML 页面在没有 WebSocket 技术的情况下,我们通常采用_____方式来实现客户端与服务器端数据的同步。

2. 在建立 WebSocket 连接时,服务器端返回报头_____同意终止 HTTP 连接,使用 WebSocket 连接。

3. 一旦建立好 WebSocket 连接,WebSocket 数据帧就可以在客户端和服务器之间以全双工模式发送和传回,_____数据帧可以同时传输。

四、简答题

简述 HTML5 WebSocket 协议的特点。

第 9 章

CHAPTER 9

播放多媒体

学习目标

- 熟练掌握< audio>以及< video>标签的使用。
- 掌握 audio 和 video 对象的常用属性。
- 掌握 audio 和 video 对象的常用方法和事件。

HTML5 标准出现之前,要在网页中播放多媒体,必须借助于 Flash 插件。使用 HTML5 提供的< audio>和< video>标签,可以很方便地在网页中播放音频和视频。本章主要介绍< audio>和< video>标签的使用,以及如何使用相应的 API 控制多媒体的播放。

9.1 HTML5 标准中的音视频

Web 开发人员曾经一直想在网页上播放多媒体,特别是当前网络的带宽速度已经很快,足以支持音视频的在线播放。在早期的技术中,原生的网页标准不支持在网页中直接嵌入音视频,所以在浏览器中需要安装插件(例如 Flash 或 SilverLight),这种技术在过去给网页增色不少,但缺点也很明显,一是无法利用 HTML/CSS 的一些特性,另外也带来了一些安全方面的问题,这在第 1 章已经详细介绍过。在 iOS 对 Flash 进行了封杀后(SilverLight 主要用在 Windows 平台上),这些技术慢慢地都会被市场淘汰。HTML5 标准中为了解决这些问题,定义了< audio>和< video>标签,以及一套 JavaScript API,实现了对音视频播放的原生支持。这两个标签的使用非常类似,只是使用的标签不同。

9.1.1 < audio>标签

< audio>标签用于播放音频,它支持的音频文件格式包括“. wav”“. mp3”和“. ogg”,如果不是这三种格式,可以使用网站“<http://cn.office-converter.com/>”提供的功能,很方便地实现音频格式的在线转换。

< audio>标签的书写格式如下:

```
<audio src="音频文件地址" id="myaudio">
  你的浏览器不支持< audio>标签
</audio>
```

如果只指定一种格式的音频文件,很有可能在某些浏览器中不能正常播放,为了能够确

保< audio>标签在各浏览器中都能实现兼容,可以在< audio>标签中再使用< source>标签加载不同格式的音频文件,浏览器会自动使用第一个能识别的格式,例如下面这段代码:

```
<audio id="myaudio">
  <source src="song.ogg" type="audio/ogg">
  <source src="song.mp3" type="audio/mpeg">
  你的浏览器不支持<video>标签
</audio>
```

9.1.2 < video>标签

< video>标签用于播放视频,它支持的视频文件格式包括“. webm”“. mp4”和“. ogg”,其中,“. mp4”一定要采用 H. 264 编码。H. 264 已经占领视频市场的 80%,如果移动应用中使用视频,建议采用 H. 264 编码,因为它具有较好的高压压缩比和画质。

< video>标签的书写格式如下:

```
<video src="音频文件地址" id="myvideo">
  你的浏览器不支持<audio>标签
</video>
```

与< audio>标签一样,在< video>标签中,也可以使用< souce>标签来实现兼容性,例如:

```
<video id="myvideo">
  <source src="song.ogg" type="video/ogg">
  <source src="song.mp4" type="video/mp4">
  你的浏览器不支持<video>标签
</video>
```

9.2 < audio>和< video>标签的主要属性

从 9.1 节可以看出,这两个标签的书写形式以及结构是基本类似的,不仅如此,除了 src 属性,它们标签中的其他基本属性也是一样的,如表 9-1 所示。

表 9-1 <audio>和<video>标签的主要属性

属性	描 述
autoplay	自动播放音频或视频
controls	显示系统自带的播放面板
loop	自动循环播放音频或视频
preload	值为“auto”表示页面载入后自动加载音频或视频;为“meta”表示页面载入后只加载音频或视频的元数据(例如音频和视频的时长);为“none”则表示页面载入后不加载音频或视频
muted	静音控制,音频或视频播放时无音量

除了上面一些共有的属性,< video>标签还有一些独有属性,如表 9-2 所示。

表 9-2 <video>标签的独有属性

属性	描 述
poster	用来定义视频播放器的预览图片
width	设置视频播放器的宽度,不需要使用单位,例如 width="320"
height	设置视频播放器的高度,不需要使用单位,例如 height="320"

如果在< audio>和< video>标签中使用了 controls 属性,例如< audio controls></audio>和< video controls></video>,则会出现如图 9-1 和图 9-2 所示的控制面板。

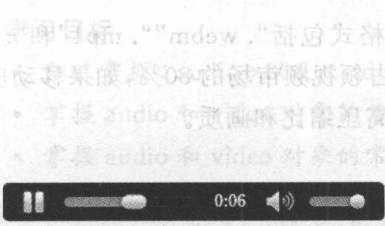


图 9-1 Chrome 自带的音频控制面板



图 9-2 Chrome 自带的视频控制面板

这里使用< audio>标签来示范这些属性的使用。

【例 9-1】< audio>标签的属性使用示范,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0,maximum - scale = 1.0, user - scalable = no" />
    <title> audio 标签的属性使用</title>
  </head>
  <body>
    <audio controls loop autoplay>
      <source src = "media/music.mp3" type = "audio/mp3"></source>
      <source src = "media/music.ogg" type = "audio/ogg"></source>
      你的浏览器不支持 auido 标签
    </audio>
  </body>
</html>
```

在 Chrome 中浏览后,会出现如图 9-1 所示的控制面板,音频会自动播放,并且在播放结束后会自动循环。



iOS 是不支持 autoplay 属性的,也不支持使用 JavaScript API 实现自动播放,必须由用户主动交互后才能播放,这是为了防止突然出现声音或者流量丢失。

9.3 audio 对象和 video 对象的 API

在 HTML5 标准中,为音频或视频对象提供了丰富的 JavaScript 控制的 API,要使用这些 API,必须先使用 DOM 查找到音视频对象,例如:

```
var myaudio = document.getElementById("myaudio");
var myvideo = document.getElementById("myvideo");
```

在这些 API 中,表 9-3 列举了音、视频对象的常用方法。

表 9-3 audio 和 video 对象的常用方法

方法	描 述
load()	重新加载音频/视频
play()	播放音频/视频
pause()	暂停当前播放的音频/视频

表 9-4 列举了音、视频对象的一些常用属性。

表 9-4 audio 和 video 对象的常用属性

属 性	描 述
autoplay	Boolean 类型,设置或返回是否在加载完成后随即播放音频/视频
buffered	返回 TimeRanges 对象,表示用户的音视频缓冲范围。缓冲范围指的是已缓冲音视频的时间范围。如果用户在音视频中跳跃播放,会得到多个缓冲范围。TimeRanges 对象属性如下: <ul style="list-style-type: none">length: 获得音视频中已缓冲范围的数量start(index): 获得某个已缓冲范围的开始位置end(index): 获得某个已缓冲范围的结束位置
currentSrc	String 类型,返回当前音频/视频的 URL,如果未设置音频/视频,则返回空字符串
currentTime	Number 类型,设置或返回音频/视频播放的当前位置(以秒计),当设置该属性时,播放会跳跃到指定的位置
duration	Number 类型,返回当前音频/视频的长度,以秒计。如果未设置音频/视频,则返回 NaN
ended	Boolean 类型,返回音频/视频是否已结束
loop	Boolean 类型,设置或返回音频/视频是否应该在结束时再次播放
muted	Boolean 类型,设置或返回音频/视频是否应该被静音(关闭声音)
paused	Boolean 类型,返回音频/视频是否已暂停
playbackRate	Number 类型,设置或返回音频/视频的当前播放速度
played	返回 TimeRanges 对象,TimeRanges 对象表示用户已经播放或看到的音频/视频范围。已播范围指的是被播放音频/视频的时间范围。如果用户在音频/视频中跳跃,则会获得多个播放范围。TimeRanges 对象属性如下: <ul style="list-style-type: none">length: 获得音频/视频中已播范围的数量start(index): 获得某个已播范围的开始位置end(index): 获得某个已播范围的结束位置
preload	Boolean 类型,设置或返回是否在页面加载后立即加载音频/视频

除了上面一些共有的属性,<video>标签还有一些独有属性,如表 9-2 所示

续表

属 性	描 述
readyState	Number 类型,返回音频/视频的当前就绪状态。0: 没有关于音频/视频是否就绪的信息; 1: 关于音频/视频就绪的元数据; 2: 关于当前播放位置的数据是可用的,但没有足够的数据来播放下一帧/毫秒;3: 当前及至少下一帧的数据是可用的;4: 可用数据足以开始播放
seekable	返回 TimeRanges 对象,TimeRanges 对象表示音频/视频中用户可寻址的范围。可寻址范围指的是用户在音频/视频中可寻址(移动播放位置)的时间范围。对于流视频,通常可以寻址到视频中的任何位置,即使其尚未完成缓冲。TimeRanges 对象的属性如下: <ul style="list-style-type: none">length: 获得音频/视频中可寻址范围的数量start(index): 获得可寻址范围的开始位置end(index): 获得可寻址范围的结束位置
volume	Number 类型,设置或返回音频/视频的当前音量,必须是介于 0.0 与 1.0 之间的数字

如果需要自定义音/视频播放器(虽然系统自带的播放器已经很好用,但还有自定义需求),这需要掌握 API 中的一些常用事件,如表 9-5 所示。

表 9-5 audio 和 video 对象的常用事件

事 件	描 述
abort	当音频/视频的加载已放弃时
canplay	当浏览器可以播放音频/视频时
canplaythrough	当浏览器可在不因缓冲而停顿的情况下进行播放时
durationchange	当音频/视频的时长已更改时
emptied	当目前的播放列表为空时
ended	当目前的播放列表已结束时
error	当在音频/视频加载期间发生错误时
loadeddata	当浏览器已加载音频/视频的当前帧时
loadedmetadata	当浏览器已加载音频/视频的元数据时
loadstart	当浏览器开始加载音频/视频时
pause	当音频/视频已暂停时
play	当音频/视频已开始或不再暂停时
playing	当音频/视频在已因缓冲而暂停或停止后已就绪时
progress	当浏览器正在下载音频/视频时
ratechange	当音频/视频的播放速度已更改时
seeked	当用户已移动/跳跃到音频/视频中的新位置时
seeking	当用户开始移动/跳跃到音频/视频中的新位置时
timeupdate	当目前的播放位置已更改时
volumechange	当音量已更改时
waiting	当视频由于需要缓冲下一帧而停止时

9.4 实例：视频播放器

前面的内容已经介绍了 HTML5 标准中为 audio 和 video 对象定义的 API 中的各属性、方法和事件。下面将实现一个自定义的播放器,给各位读者作为参考。

【例 9-2】自定义的视频播放器。

使用 Chrome 打开页面文件后,打开页面的效果如图 9-3 所示,单击播放按钮后,运行的效果如图 9-4 所示。



图 9-3 Chrome 打开页面的效果



图 9-4 视频播放效果

本例中的视频播放器实现了以下自定义功能:播放或暂停、全屏、静音控制、鼠标拖放进度,播放时控制面板会自动隐藏,该控制面板并非浏览器原生视频自带的,而是通过 HTML 和 CSS 进行模拟定制的。

由于相应的代码较多,就不在此处赘述了,请参考本书的配套源代码。

视频播放器的脚本控制中使用了 video 对象的 play() 和 pause() 方法来播放或暂停;使用了 muted 属性来控制静音;使用了 loadedmetadata 事件来显示视频时长;使用了 timeupdate 事件来实现视频播放的进度条;使用了 seeking 事件来处理视频跳转;使用了 waiting 事件来处理视频的待播放;使用了 ended 事件来处理视频播放的结束:这都是目前视频播放器常用的一些功能。可以看出,结合 HTML 和 CSS,制作出一个界面精美、功能强大的视频播放器并非难事。

小结

本章介绍了 HTML5 标准中的< audio >和< video >标签的使用,以及所对应的 audio 和 video 对象的各 API,并在最后使用这些 API 实现了一个视频播放器的实例。在移动互联网时代,众多的浏览器都已不再依赖 Flash 插件,相信熟练掌握 audio 和 video 对象的 API,就可以打造出超具个性的音视频播放器。

习题

一、选择题

1. < audio >标签支持的音频文件类型不包括()。
A. .wav B. .mp3 C. .ogg D. .aud
2. < video >标签支持的视频文件类型不包括()。
A. .mp4 B. .webm C. .flv D. .ogg
3. < audio >和< video >标签中用于控制自动循环播放的属性是()。
A. controls B. loop C. autoplay D. preload
4. 控制视频或音频暂停应该使用的方法是()。

- A. stop()

B. pause()

C. paused()

D. play()
5. 用于设置或返回音视频当前播放时间的 audio 或 video 对象的属性是()。
- A. currentTime

B. time

C. playTime

D. currentPlayTime

二、判断题

1. <video>标签支持所有的. mp4 格式的视频播放。()
2. <audio>或<video>标签都可以在内部使用多个<source>标签来引用不同格式的文件。()
3. 移动互联网时代的浏览器不再依赖 Flash 插件就可以播放多媒体了。()
4. audio 和 video 对象 API 中的 timeupdate 事件只是加载视频时执行一次。()

三、填空题

1. 在 HTML5 标准中,使用_____标签定义一个音频播放器,使用_____标签定义一个视频播放器。
2. <video>标签中使用 poster 属性定义_____。
3. <audio>和<video>标签的_____属性用于定义是否显示控制面板。
4. audio 和 video 对象可以使用_____属性控制音视频对象的播放速度。
5. 可以使用 video 对象的_____事件,来处理当播放视频时视频缓冲下一帧时而暂停的逻辑。

第 10 章

CHAPTER 10

本地存储

学习目标

- 了解 HTML5 的本地存储技术。
- 熟练掌握 localStorage 和 sessionStorage 的使用。
- 掌握 Web SQL 数据库的使用。
- 掌握 IndexedDB 数据库的使用。

传统 Web 应用程序的本地存储能力较弱,主要以 Cookie 方式存储。HTML5 的本地存储能力得到了较大提高,不但可以像传统 Web 应用程序那样实现本地数据简单存储,还可以支持本地的轻型数据库。本章介绍 HTML5 开发中的 localStorage、sessionStorage、Web SQL 数据库、IndexedDB 数据库这 4 种本地存储技术,在 HTML5 App 开发中可以灵活选用。

10.1 HTML5 本地存储技术概述

本地存储技术由来已久,使用它可以减少向服务器的请求次数,从而减少用户等待从服务器端获取数据的时间;同时,在网络状态不佳时,仍可以显示离线数据。

传统的 Web 应用中主要采用 Cookie 实现本地存储,Cookie 是由 Web 服务器保存在用户浏览器上的小文本文件,它包含有关用户的信息,会跟随 HTTP 请求一起发送。无论何时用户链接到服务器,Web 站点都可以访问 Cookie 信息。但是 Cookie 只能存储文本信息,有时间限制,会增加网络流量,另外最多也只能存储 4KB 的数据。在当前的移动互联网时代,特别是在 HTML5 App 开发中并不适用,例如使用 Cookie 来存储用户编辑文档时的草稿就是一个问题。

HTML5 标准大大扩充了本地存储的能力,它提供了以下几种新增的本地存储技术:

1) localStorage

localStorage 类似于 Cookie,用于持久化本地存储,但它没有时间限制,除非主动删除和清空内容,否则数据永不过期。另外,它的存储能力也远大于 Cookie,可以存储多达 5MB 的数据。

2) sessionStorage

sessionStorage 类似于 Session,用于本地存储一个会话(session)中的数据。这些数据只有在同一个会话中的页面才能访问,当会话结束后(关掉网站所有浏览的页面或浏览器),

数据也会随之丢失。因此,sessionStorage 不是一种可持久化的本地存储。

3) Web SQL 数据库

localStorage 和 sessionStorage 这两个是以键值对存储的解决方案,用于存储少量数据结构很有用,但是对于大量结构化数据就无能为力了。我们经常在数据库中处理大量结构化数据,HTML5 引入 Web SQL 数据库概念,它使用 SQL 来操纵客户端数据库的 API,这些 API 是异步的,规范中使用的是 SQLite。SQLite 是一款轻型的数据库,是遵循 ACID (原子性、一致性、隔离性、持久性)的关系型数据库管理系统。它的设计目标是嵌入式的,它占用资源非常低,只需要几百 K 字节的内存就可以了,另外,它的处理速度很快。

4) IndexedDB 数据库

IndexedDB 是一种轻量级 NoSQL(Not Only SQL)数据库,NoSQL 数据库是非关系性的数据库,它不需要使用 SQL 语句去操作数据库,数据的形式采用 JSON 格式,提供了常规的 CRUD(增查改删)操作,并支持事务,所有操作都是异步的,储存空间也比较大。

10.2 localStorage 和 sessionStorage

localStorage 和 sessionStorage 这两种存储对象具有相同的属性和方法,但 localStorage 对象是持久化存储,没有时间限制,而 sessionStorage 是非持久化存储,当关闭了页面或浏览器,则会销毁数据。所有最新版本的浏览器均支持这两个存储特性,表 10-1 列出了当前浏览器的支持情况。

表 10-1 主流的浏览器对 localStorage 和 sessionStorage 支持情况

浏 览 器	支持情况
Chrome	4.0 及以上版本
FireFox	4.0 及以上版本
Internet Explore	8.0 及以上版本
Opera	11.0 及以上版本
Safari	4.0 及以上版本
iOS	5.0 及以上版本
Android	3.0 及以上版本

10.2.1 检查浏览器的支持

在使用 localStorage 和 sessionStorage 这两种存储对象存储数据时,最好使用下面的代码检查浏览器的支持情况:

```
if(window.localStorage){
    //实现存储
}
else{
    alert("你的浏览器不支持 localStorage");
}
```

```

if(window.sessionStorage){
    //实现存储
}
else{
    alert("你的浏览器不支持 sessionStorage");
}

```

10.2.2 相应的 API

localStorage 和 sessionStorage 是使用键值对进行数据存储、通过键值进行检索的，表 10-2 显示了可用的 API 方法。



实际使用中，localStorage 和 sessionStorage 保存 JavaScript 对象都是将其序列化成字符串，检索出来后再反序列化。

表 10-2 localStorage 和 sessionStorage 的 API 方法

方 法	说 明
setItem(key, value)	为 Web 存储对象添加一个键/值对，供以后使用。该值可以是任何的数据类型：字符串、数值、数组等
getItem(key)	基于起初用来存储它的这个键检索值
clear()	清除所有的键/值对数据
removeItem(key)	基于某个键从此 Web 存储对象清除特定的键/值对
key(n)	检索第 n 个键的值

【例 10-1】 localStorage 使用示例，步骤如下。

① 新建 Web 项目后，在项目中新建 HTML 文件“l1.html”，页面代码如下：

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title>localStorage 示范</title>
        <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    </head>
    <body>
        <input type = "text" id = "myvalue"/>
        <input type = "button" id = "btnSave" value = "存储数据并跳转"/>
        <script>
            document.getElementById('btnSave')
                .addEventListener('click',function () {
                var val = document.getElementById("myvalue").value;
                //如果支持 localStorage
                if(window.localStorage){

```

```

        //localStorage 保存输入值并跳转到 12.html
        localStorage.setItem("myval", val);
        location.href = "12.html";
    }
    });
</script>
</body>
</html>

```

② 在项目中新建 HTML 文件“12.html”，页面代码如下：

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <title></title>
    <title> localStorage 示范</title>
    <meta name = "viewport"
content = "initial-scale = 1.0, maximum-scale = 1.0, user-scalable = no" />
  </head>
  <body>
    <input type = "button" value = "读取数据" id = "btnRead"/>
    <input type = "button" value = "删除数据" id = "btnRemove"/>
    <input type = "button" value = "清空数据" id = "btnClear"/>
    <script>
      document.getElementById('btnRead')
        .addEventListener('click',function () {
          //若支持 localStorage
          if(window.localStorage){
            //根据键"myval"取值
            alert(localStorage.getItem("myval"));
          }
        });
      document.getElementById('btnRemove')
        .addEventListener('click',function () {
          //若支持 localStorage
          if(window.localStorage){
            //根据键"myval"删除值
            localStorage.removeItem("myval");
          }
        });
      document.getElementById('btnClear')
        .addEventListener('tap',function () {
          //若支持 localStorage
          if(window.localStorage){
            //清空 localStorage 中的数据
            localStorage.clear();
          }
        });
    </script>
  </body>
</html>

```



```

    });
  </script>
</body>
</html>

```

③ 在 Chrome 浏览器中浏览“l1.html”，在页面中显示一个文本输入框和一个按钮，如图 10-1 所示，输入数据（例如“hello html5”）后，单击“存储数据并跳转”按钮，页面自动跳转到“l2.html”，在“l2.html”中单击“读取数据”会自动弹出“l2.html”。如果单击“删除数据”按钮，再单击“读取数据”按钮，则会弹出“null”值，单击“清空数据”后，所有的数据自动销毁，效果如图 10-2 所示。

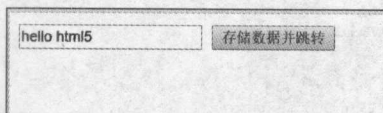


图 10-1 l1.html 页面效果

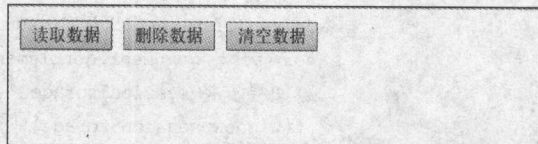


图 10-2 l2.html 页面效果

④ 再次在 Chrome 浏览器中浏览“l1.html”，输入数据并单击按钮后，打开 Chrome 的“开发者工具”，选择“Application”面板，在左边的树型菜单中选择“Storage”下面的“Local Storage”选项，在它下面列出了以网站域名作为分类、在本机上存储的所有 localStorage 的键/值对，如图 10-3 所示，在本例中使用的键“myval”存储值“hello html5”显示在了右边。

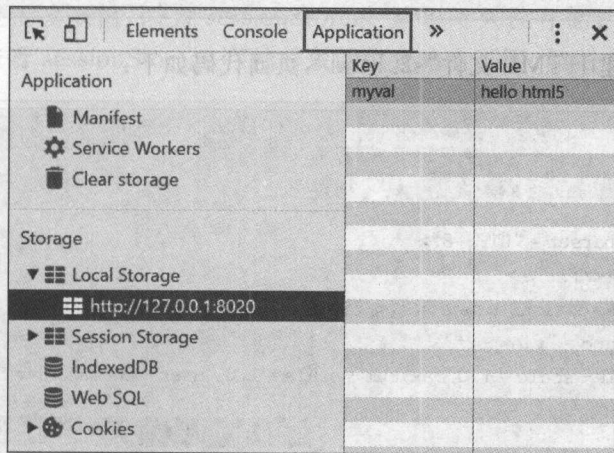


图 10-3 Chrome 中 localStorage 对象存储的查看

⑤ 关闭 Chrome 浏览器后，再次打开 Chrome，直接浏览“l2.html”，再单击“读取数据”按钮，依然可以得到刚才存储的值“hello html5”，这说明 localStorage 是持久性存储。

【例 10-2】 sessionStorage 使用示例，步骤如下。

① 新建 Web 项目后，在项目中新建 HTML 文件“s1.html”，页面代码如下：

```
<!DOCTYPE html>
```

```

<html>
  <head>
    <meta charset = "UTF - 8">
    <title>sessionStorage 示范</title>
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
  </head>
  <body>
    <input type = "text" id = "myvalue" />
    <input type = "button" id = "btnSave" value = "存储数据并跳转" />
    <script>
      document.getElementById('btnSave')
        .addEventListener('click', function() {
        var val = document.getElementById("myvalue").value;
        //如果支持 sessionStorage
        if(window.sessionStorage) {
          //sessionStorage 保存输入值并跳转到 s2.html
          sessionStorage.setItem("myval", val);
          location.href = "s2.html";
        }
      });
    </script>
  </body>
</html>

```

② 在项目中新建 HTML 文件“s2.html”，页面代码如下：

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <title></title>
    <title>sessionStorage 示范</title>
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
  </head>
  <body>
    <input type = "button" value = "读取数据" id = "btnRead" />
    <input type = "button" value = "删除数据" id = "btnRemove" />
    <input type = "button" value = "清空数据" id = "btnClear" />
    <script>
      document.getElementById('btnRead')
        .addEventListener('click', function() {
        //若支持 sessionStorage
        if(window.sessionStorage) {
          //根据键"myval"取值

```

```

        alert(sessionStorage.getItem("myval"));
    }
});
document.getElementById('btnRemove')
    .addEventListener('click', function() {
        //若支持 sessionStorage
        if(window.sessionStorage) {
            //根据键"myval"删除值
            sessionStorage.removeItem("myval");
        }
    });
document.getElementById('btnClear')
    .addEventListener('tap', function() {
        //若支持 sessionStorage
        if(window.sessionStorage) {
            //清空 sessionStorage 中的数据
            sessionStorage.clear();
        }
    });
</script>
</body>
</html>

```

这个例子的效果和例 10-1 类似,不同的是:如果关闭 Chrome 后,再直接浏览“s2.html”会发现无法直接读取值,因为 sessionStorage 是非持久性存储;另外,在 Chrome 的“开发者工具”中,查看 sessionStorage 存储对象的位置略有不同,如图 10-4 所示。

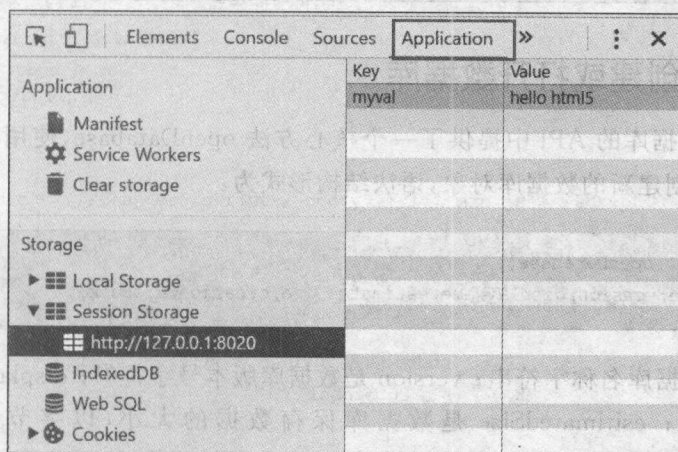


图 10-4 Chrome 中 sessionStorage 对象存储的查看



HTML5 手机 App 中主要使用 localStorage, 在 Web App 或网页中视情况也可用 sessionStorage。

10.3 Web SQL 数据库

在 W3C 的 Web SQL 数据库规范中引入了一套使用 SQL 来操纵客户端数据库(Client-Side Database)的 API,这些 API 是异步的,规范中所使用的 SQL 语言为 SQLite 3.6.19。Web SQL 并未纳入 HTML5 标准中,所以不是所有浏览器都支持它,表 10-3 显示了目前各浏览器对 Web SQL 数据库的支持情况。

表 10-3 主流的浏览器对 Web SQL Database 的支持情况

浏 览 器	支 持 情 况
Chrome	42.0 及以上版本
FireFox	不支持
Internet Explore、Edge	不支持
Opera	42.0 及以上版本
iOS Safari	9.3 和 10.1
Android Browser	2.1 及以上版本

根据 window 对象的 openDatabase 属性是否存在,可以检测出浏览器内核是否支持 Web SQL 数据库,例如下面的代码:

```
if(window.openDatabase){  
    //支持,可以处理数据库代码  
}
```

10.3.1 创建或打开数据库

Web SQL 数据库的 API 中提供了一个核心方法 openDatabase,使用它可以使用现有的数据库对象或创建新的数据库对象,语法结构形式为:

```
var db = window.openDatabase(  
    name,version,displayName,estimatedSize,creationCallback);
```

其中: name 是数据库名称字符串; version 是数据库版本号字符串; displayName 是数据库显示名称字符串; esitimatedSize 是数据库保存数据的大小,以字节为单位的数值; creationCallback 是创建或连接数据库成功后的回调函数。

执行后,可以通过 db 对象检测创建或连接数据库是否成功。

10.3.2 执行 SQL 语句

Web SQL 数据库的 API 使用 transaction 方法执行 SQL 语法,transaction 方法用以处理事务,当一条语句执行失败的时候,整个事务回滚。它的语法形式如下:

```

db.transaction(function(tx){
    tx.executeSql(sqlString,
        paramArray,
        successCallback,
        failedCallback);
});

```

API 在这里是使用 executeSql 方法控制 SQL 语句的执行,它有 4 个参数: sqlString 是要执行的 SQL 语句字符串; paramArray 是要传递的参数数组,若没有参数,则使用[]; successCallback 是成功回调函数,一般用于处理查询结果集; failedCallback 是出现错误的回调函数。

【例 10-3】 Web SQL Database 使用示例,步骤如下。

① 新建 Web 项目后,创建 HTML 页面,页面代码如下:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
        <title>Web SQL Database 示例</title>
        <style>
            /* 样式省略,自行参看示例 */
        </style>
    </head>
    <body>
        <div class="title">收货地址</div>
        <table id="dataTable">
            <tbody>
            </tbody>
        </table>
    </body>
    <script src="../../Chapter6/scripts/jquery-3.1.1.min.js"></script>
    <script>
        var App = null;
        $(function() {
            App = {
                $tbody: $("tbody"), //表格的 tbody
                db: null             //indexedDB 数据库对象
            };
            if(window.openDatabase) {
                App.db = openDatabase('mydb', '1.0', 'Test DB', 100);
                if(App.db) {
                    //创建 Address_info 表并插入 3 条数据,如果数据表已存在则跳过,由事务保证
                    App.db.transaction(function(tx) {
                        tx.executeSql('CREATE TABLE IF NOT EXISTS
Address_info (No unique, Receiver,Tel,Address)');

```

```

        tx.executeSql("INSERT INTO Address_info(No,Receiver,
Tel,Address) VALUES(1,'黄波','13611111111','成都天府广场 1 号')");
        tx.executeSql("INSERT INTO Address_info(No,Receiver,
Tel,Address) VALUES(2,'张小华','13622222222','成都天府广场 2 号')");
        tx.executeSql("INSERT INTO Address_info(No,Receiver,
Tel,Address) VALUES(3,'黄平','13633333333','成都天府广场 3 号')");
    });
    //读取数据,生成表格
    App.db.transaction(function(tx){
        tx.executeSql("SELECT * FROM Address_info",[],
        function(tx,result){
            for(var i = 0;i<result.rows.length;i++){
                var $tr = $("<tr></tr>");
                var $td1 = $("<td></td>");
                $td1.text(result.rows[i].Receiver);
                $tr.Append($td1);
                var $td2 = $("<td></td>");
                $td2.text(result.rows[i].Tel);
                $tr.Append($td2);
                var $td3 = $("<td></td>");
                $td3.text(result.rows[i].Address);
                $tr.Append($td3);
                App.$tbody.Append($tr);
            }
        });
    });
}
});
</script>
</html>
```

② 这个例子在 Chrome 中浏览的效果如图 10-5 所示,程序创建了一个数据库“mydb”,并在数据库中新建了表“Address_info”,并插入了 3 条数据。

收货地址		
黄波	13611111111	成都天府广场1号
张小华	13622222222	成都天府广场2号
黄平	13633333333	成都天府广场3号

图 10-5 Web SQL Database 示例效果

③ 打开 Chrome 的“开发者工具”后,选择“Application”面板,在左边的树形菜单中选择“Storage”下面的“Web SQL”选项,如图 10-6 所示,可以看到 Web SQL 数据库存储的情况。

10.4.1 数据库初始化

创建或打开数据库都使用 open 方法,这个方法的语法如下:

```
var req = indexedDB.open(name, version);
```

其中,name 是数据库名称字符串,version 是数据库版本号字符串。

如果指定的数据库存在,则打开,否则创建数据库。IndexedDB 数据库的操作完全是异步进行的,每一次 IndexedDB 数据库操作,都需要定义操作在成功或失败的回调函数,方法如下:

```
if(window.indexedDB) {  
    var req = indexedDB.open(name, version);  
    var db;  
    req.onsuccess = function(event) {  
        db = event.target.result;  
    };  
    req.onerror = function(error) {  
        console.log(err.target.errorCode);  
    };  
    req.onupgradeneeded = function(event) {  
        db = event.target.result;  
    };  
}
```

在这里,db 代表创建或打开数据库后获得的数据库实例,通过实例可以对数据库作后续操作,onupgradeneeded 会在创建数据库或数据库版本号升高时自动触发,onsuccess 是创建或打开数据库成功后的回调函数,onupgradeneeded 会在 onsuccess 之前调用,onerror 是打开或创建失败时的回调函数。

10.4.2 对象存储空间

IndexedDB 不是关系数据库,它使用对象存储空间(ObjectStore)来存储数据,它类似于关系数据库中的表。一个数据库中 can 包含多个对象存储空间,对象存储空间使用键值对的形式来存储数据,即每个数据都由一组键和一组值组成,值是一个 JSON 对象。

使用数据库实例对象的 createObjectStore 方法可以创建对象存储空间,方法如下:

```
var store = db.createObjectStore(name, optionalParameters);
```

其中,name 是对象存储空间名字字符串,后面的参数可选,代表对象存储空间中键值的选项,是一个 JSON 对象配置。我们可以使用每条记录中的某个指定字段作为键值(keyPath),也可以使用自动生成的递增数字作为键值(keyGenerator),也可以不指定。选择键的类型不同,objectStore 可以存储的数据结构也有差异,它们的组合含义如表 10-5 所示。

表 10-5 objectStore 键的选项

键 类 型	描 述
不使用	可以存储任意值,但是没添加一条数据的时候需要指定键参数
keyPath	只能存储 JavaScript 对象,存储对象的一个属性和键值相同
keyGenerator	可存储任意值,当保存新值时,可以自动生成键,相当于自增长列
keyPath 和 keyGenerator	只能存储 JavaScript 对象,存储对象的一个属性和键值相同,当保存新值时,可以自动生成键

数据库实例对象的 objectStoreNames 属性中包含了数据库所有的对象存储空间名,在创建对象存储空间之间,最好先判断要创建的对象存储空间名称是否已经存在,例如:

```
req.onupgradeneeded = function(event) {
    db = event.target.result;
    //判断对象存储空间是否存在,没有就创建
    if(!db.objectStoreNames.contains("test")){
        //创建对象存储空间,键值是属性 No
        db.createObjectStore("test", {keyPath: "No"});
    }
};
```

10.4.3 索引

数据库中的索引是一个表(对象存储空间)中所包含的值的列表,当 IndexedDB 数据库需要使用其他属性(非主键)获取数据时,就要预先创建索引,然后使用索引获取数据。

可以通过调用 ObjectStore 对象的 createIndex 方法在对象存储空间中创建索引,方法如下:

```
store.createIndex(name, keyPath, optionalParameters);
```

其中: name 是索引名字符串; keyPath 是要创建索引的对象属性字符串; optionalParameters 是设置索引是否唯一的 JSON 对象,一般都设置为 {unique: true}。

10.4.4 事务

数据库中的事务是包含一组数据库操作的逻辑工作单元,在事务中包含的数据库操作是不可分割的整体,要么一起执行,要么回滚到执行事务之间的状态。IndexedDB 数据库中常用的对数据库的操作(例如插入、删除、更改数据)都需要在事务里完成。

创建事务使用 transaction 方法,它的方法如下:

```
db.transaction(storeName, mode);
```

其中: storeName 是要操作的对象存储空间名字符串; mode 是事务模式字符串,可以使用“readonly”(只读模式)或“readwrite”(可读写模式),若不指定,默认为只读模式。

事务对象可以配置发生错误、终止和事务中所有操作都完成的回调函数,例如:

```
var trans = db.transaction("test", "readwrite");
trans.onerror = function(){
    //事务发生错误的处理
};
trans.onabort = function(){
    //事务被终止的处理
};
trans.oncomplete = function(){
    //事务操作全部完成后的处理
};
```

10.4.5 IndexedDB 的 CRUD 操作

数据库操作都包含了增加(Create)、读取(Retrieve)、更改(Update)、删除(Delete)这几项,IndexedDB 数据库中是使用对象存储空间对象的相应的方法,例如:

```
//获取相应的对象存储空间
var store = db.transaction("test", "readwrite")
                .objectStore("test");

//增加数据,data 是 JSON 对象
store.add(data);

//读取数据,keyPath 是键值
store.get(keyPath).onsuccess = function(){
    //更新数据,data 是新的 JSON 对象
    store.put(newdata);
};

//删除数据,keyPath 是键值
store.delete(keyPath);
```

10.4.6 游标

通过对象存储空间对象的 get 方法只能根据键值 keyPath 从对象存储空间中获取数据,如果要取得对象存储空间中的一组数据,就需要使用游标。游标是映射在结果集中一行数据上的位置实体,有了游标,用户就可以访问结果集中的任意一行数据了。最常见的就是遍历所有数据,用法如下:

```
var store = db.transaction("test").objectStore("test");
store.openCursor().onsuccess = function(e) {
    var cur = e.target.result;
    var datas = [];
    if(cur) {
```

```

datas.push(cur.value);    //每一条数据
cur.continue();           //下一条
}

```

【例 10-4】 IndexedDB 使用示例。

在 Chrome 中浏览后,运行效果如图 10-7 所示,本例示范了使用 IndexedDB 技术创建数据库“Info”和对象存储空间“address_info”,并添加了 3 条对象数据、修改数据、遍历数据,为对象存储空间设定索引“Tel”,根据索引查询数据,根据键值“No”属性查询数据,删除数据库等功能。

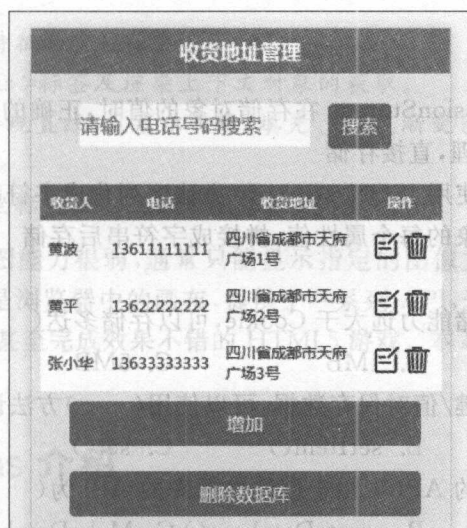


图 10-7 IndexedDB 示例效果

打开 Chrome 的“开发者工具”后,选择“Application”面板,在左边的树形菜单中选择“Storage”下面的“IndexedDB”选项,如图 10-8 所示,可以看到 IndexedDB 数据库存储的情况。

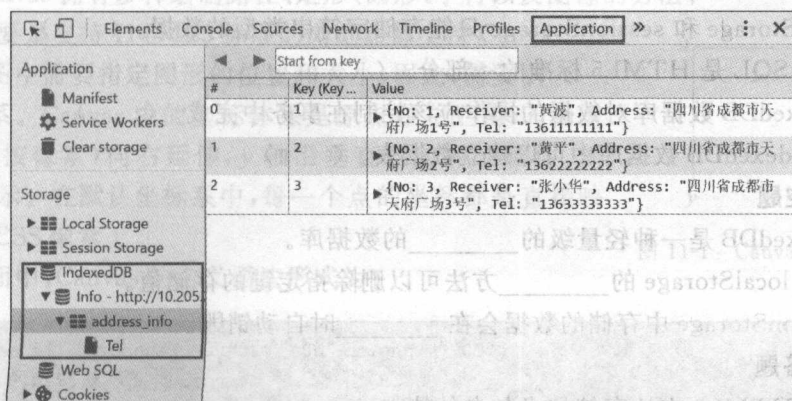


图 10-8 Chrome 中 IndexedDB 数据库的存储

小结

本章介绍了 HTML5 App 开发中常用的 4 种本地存储技术,先是介绍了 localStorage 和 sessionStorage 的使用,它们主要以键/值对的形式存储字符串数据;然后讲解了 Web SQL 数据库的使用;最后讲解了 IndexedDB 数据库的使用,并给出了一个 IndexedDB 的综合实例。HTML5 的本地存储能力得到了很大提高,在实际的开发中,可以根据项目的复杂度对这 4 项技术进行选择。

习题

一、选择题

1. localStorage 和 sessionStorage 在存储对象的值时,正确的处理方式应为()。
A. 不需要任何处理,直接存储
B. 将 JSON 对象使用 JSON.stringify 方法序列化后成字符串,再将其保存
C. 取出 JSON 对象的每个属性值,拼接成字符串后存储
D. 没办法存储
2. localStorage 的存储能力远大于 Cookie,可以存储多达()的数据。
A. 100k B. 1MB C. 5MB D. 10MB
3. localStorage 使用键/值对保存数据,可以使用()方法设置 localStorage 数据。
A. Save() B. setItem() C. set() D. Insert()
4. Web SQL 数据库的 API 中用于创建数据库的 API 为()。
A. newDatabase() B. createDatabase() C. MakeDatabase() D. openDatabase()
5. 在 IndexedDB 数据库中,通过对象存储空间对象的()方法可以向对象存储空间中添加数据。
A. insert() B. Append() C. insertinto() D. add()

二、判断题

1. sessionStorage 实现的是非持久性存储。()
2. localStorage 和 sessionStorage 只能存储字符串类型的数据。()
3. Web SQL 是 HTML5 标准的一部分。()
4. IndexedDB 数据库对数据的操作应该控制在事务中完成。()
5. 在 IndexedDB 数据库中建立数据表。()

三、填空题

1. IndexedDB 是一种轻量级的_____的数据库。
2. 调用 localStorage 的_____方法可以删除指定键的存储值。
3. sessionStorage 中存储的数据会在_____时自动销毁。

四、简答题

请简述 HTML5 本地存储技术各自的特点。

第 11 章

CHAPTER 11

Canvas 绘图

学习目标

- 了解 Canvas 的特征和坐标体系。
- 熟练掌握<canvas>标签及渲染上下文对象的获取。
- 掌握使用 API 实现直线、贝塞尔曲线、填充、矩形、渐变色、圆弧、文字、图片、擦除等绘制。
- 掌握 Canvas 的坐标变换以及像素操作。

原有的网页标准绘图能力很弱,通常只能显示指定的图像文件。HTML5 标准提供了一个<canvas>标签,它是浏览器中的画布,提供了一系列 API,开发人员加以利用就可以绘制各种图形、文字、图片,甚至完成效果不错的 HTML5 游戏。本章主要针对 Canvas 的各种绘图 API 作详细的讲解。

11.1 Canvas 介绍

Canvas 是 HTML5 标准中为浏览器定义的一个画布,主要用于图形表示、图表绘制、游戏制作,它有如下特征:

- Canvas 像传统的银幕,是个矩形;
- 使用 JavaScript 在 Web 上绘制各种图像;
- Canvas 区域中的每个像素都可控,即像素级操作;
- Canvas 拥有多种绘制路径、矩形、圆形、字符以及图像的方法;
- 只要是支持 HTML5 标准的浏览器都支持 Canvas。

在绘图中需要指定图形的位置和大小,因此,需要引入一个坐标体系。Canvas 中的坐标体系是原点在左上角, x 轴沿水平方向(按像素)向右延伸, y 轴沿垂直方向向下延伸,如图 11-1 所示。在默认坐标系中,每一个点的坐标都是直接映射到一个 CSS 像素上。

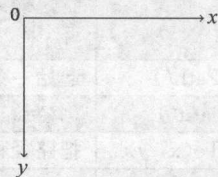


图 11-1 Canvas 的坐标体系

在页面中,Canvas 的定义语法形式为:

```
<canvas id = "mycanvas" width = "400" height = "300">
  你的浏览器不支持 Canvas.
</canvas>
```

其中,width 和 height 分别表示 Canvas 的宽度和高度,默认为像素,不需要使用单位“px”,包括使用 JavaScript 动态修改这两个属性时也是如此,直接赋值就可以。当浏览器不支持 Canvas 时,< canvas >和</ canvas >标签之间定义的内容会自动显示。



< canvas >标签的宽度和高度不能使用 CSS 定义,否则绘制的图形会被拉伸变形。

< canvas >标签对象本身是没有绘图能力的,真正要实现绘图必须使用 Canvas 对象的各种 API,在使用前,必须获得它的渲染上下对象 CanvasRenderingContext2D,取得这个对象以后,才能调用各 API 实现在 Canvas 中绘图,这就类似于每一张画布都对应一支画笔,要想在画布上绘画,就先要拿到对应的画笔,然后使用这支画笔在画布上绘图。获取渲染上下文对象的代码如下:

```
var ctx = document.getElementById("myCanvas").getContext("2d");
```

下文中,如果没有特别说明,ctx 变量都代表 CanvasRenderingContext2D 对象。

11.2 绘制图形

11.2.1 绘制直线

在渲染上下文对象的 API 中,与绘制直线相关的属性和方法见表 11-1。

表 11-1 绘制直线相关 API

属性或方法	说 明
strokeStyle	用于设置画笔绘制路径的颜色、渐变和模式。该属性的值可以是一个表示 css 颜色值的字符串。如果绘制需求比较复杂,该属性的值还可以是一个 CanvasGradient 对象或者 CanvasPattern 对象
lineWidth	定义绘制线条的宽度。默认值是 1.0,并且这个属性必须大于 0.0。较宽的线条在路径上居中,每边各有线条宽的一半
globalAlpha	定义绘制内容的透明度,取值在 0.0(完全透明)和 1.0(完全不透明)之间,默认值为 1.0
lineCap	指定线条两端的线帽如何绘制。合法的值是“butt”“round”和“square”。默认值是“butt”
beginPath()	起始一条路径,或重置当前路径
closePath()	创建从当前点回到起始点的路径
moveTo(x, y)	把路径移动到画布中的指定点(x,y),不创建线条
lineTo(x,y)	添加一个新点,然后在画布中创建从该点到最后指定点的线条
stroke()	绘制已定义的路径

在 Canvas 的图形绘制过程中,几乎都是先按照一定的顺序确定几个坐标点,也就是所谓的绘制路径,然后再根据需要,将这些坐标点用指定的方式连接起来,就形成了我们所需要的图形。当我们了解了 CanvasRenderingContext2D 对象的上述 API 后,那么绘制线条就显得非常简单了。

【例 11-1】 使用 Canvas API 绘制直线,代码如下:

```

<!DOCTYPE html >
<html >
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title> Canvas API 绘制直线</title>
  </head>
  <body>
    <canvas id = "mycanvas" width = "400" height = "400">
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
        .getContext("2d");          //获取渲染上下文对象
      //绘制封闭的三角形
      ctx.strokeStyle = "red";        //设置线条颜色
      ctx.lineWidth = 2;              //设置线条宽度
      ctx.beginPath();               //开始绘图路径
      ctx.moveTo(30,30);             //移动画笔到(30,30)
      ctx.lineTo(100,200);           //连线到(100,200)
      ctx.lineTo(30,200);            //连线到(30,200)
      ctx.closePath();               //关闭绘制路径
      ctx.stroke();                  //绘制

      //绘制两条颜色不同的线条
      ctx.beginPath();
      ctx.moveTo(280,30);
      ctx.lineTo(350,200);
      ctx.closePath();
      ctx.stroke();
      ctx.strokeStyle = "blue";
      ctx.beginPath();
      ctx.moveTo(350,200);
      ctx.lineTo(280,200);
      ctx.closePath();
      ctx.stroke();
    </script>
  </body>
</html>

```

在 Chrome 中浏览该页面的结果如图 11-2 所示,特别是在右侧图中,为了实现两条颜色不同的线条,必须使用 beginPath 和 closePath 方法实现两个不同的路径。

【例 11-2】 使用 Canvas API 绘制复杂图形,代码如下,页面浏览后效果如图 11-3 所示。


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title>Canvas API 绘制复杂图形</title>
  </head>
  <body>
    <canvas id = "mycanvas" width = "400" height = "400">
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
        .getContext("2d"); //获取渲染上下文对象

      var dx = 150;
      var dy = 150;
      var s = 100;
      var dig = Math.PI/15 * 11;
      ctx.moveTo(dx,dy);
      ctx.beginPath();
      for(var i = 0; i < 30; i++){
        var x = Math.sin(i * dig);
        var y = Math.cos(i * dig);
        //计算顶点
        ctx.lineTo(dx + x * s, dy + y * s);
      }
      ctx.closePath();
      ctx.stroke();
    </script>
  </body>
</html>
```

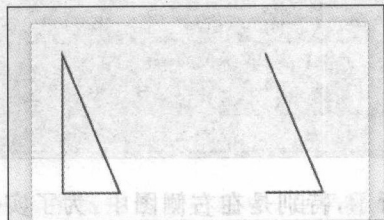


图 11-2 canvas 绘制直线效果

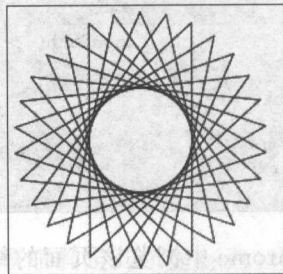


图 11-3 canvas 绘制复杂图形

11.2.2 绘制贝塞尔曲线

贝塞尔曲线是计算机图形图像造型基本工具,如图 11-4 所示,是图形造型运用得最多的基本线条之一。它通过控制曲线上的 4 个点(起始点、终止点以及两个相互分离的控制端点)来创造、编辑图形。其中,起重要作用的是位于曲线中央的控制线。这条线是虚拟的,中间与贝塞尔曲线交叉,两端是控制端点。移动两端的端点时,贝塞尔曲线改变曲线的曲率(弯曲的程度);移动中间点(也就是移动虚拟的控制线)时,贝塞尔曲线在起始点和终止点锁定的情况下做均匀移动。有兴趣的读者可以访问下面这个网址: <http://cubic-bezier.com/#.17,.67,.83,.67>,用鼠标拖拉两个控制点直观感受一下。Canvas 可以绘制两种贝塞尔曲线:二次贝塞尔曲线和三次贝塞尔曲线。

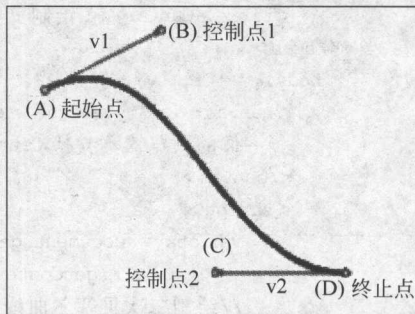


图 11-4 贝塞尔曲线示意图

二次贝塞尔曲线的路径由 3 个点确定,可以通过渲染上下文的 `quadraticCurveTo` 方法进行绘制,它的语法如下:

```
ctx.quadraticCurveTo(cpx, cpy, x, y);
```

其中, `cpx` 和 `cpy` 分别表示控制端点的 x 坐标和 y 坐标, `x` 和 `y` 表示终止点的 x 坐标和 y 坐标。

三次贝塞尔曲线的路径由 4 个点确定,可以通过渲染上下文的 `bezierCurveTo` 方法进行绘制,它的语法如下:

```
ctx.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y);
```

其中, `cp1x` 和 `cp1y` 分别表示第一个控制端点的 x 坐标和 y 坐标, `cp2x` 和 `cp2y` 分别表示第二个控制端点的 x 坐标和 y 坐标, `x` 和 `y` 分别表示终止点的 x 坐标和 y 坐标。

【例 11-3】 用 Canvas API 分别绘制二次贝塞尔曲线和三次贝塞尔曲线,要实现的效果如图 11-5 和图 11-6 所示。

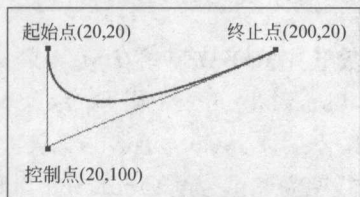


图 11-5 二次贝塞尔曲线示意

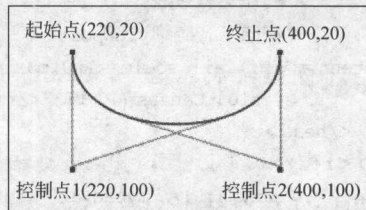


图 11-6 三次贝塞尔曲线示意

实现的代码如下:

```
<!DOCTYPE html>
```

```

<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title>Canvas API 绘制贝塞尔曲线</title>
  </head>
  <body>
    <canvas id = "mycanvas" width = "400" height = "400">
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
        .getContext("2d"); //获取渲染上下文对象
      //绘制二次贝塞尔曲线
      ctx.moveTo(20, 20);
      ctx.quadraticCurveTo(20, 100, 200, 20);
      //绘制三次贝塞尔曲线
      ctx.moveTo(220, 20);
      ctx.bezierCurveTo(220, 100, 400, 100, 400, 20);
      ctx.stroke();
    </script>
  </body>
</html>

```

11.2.3 绘制填充

CanvasRenderingContext2D 的 API 中实现填充的 API 比较简单, 可以使用 fillStyle 来设置填充颜色, 和 strokeStyle 很类似, 再使用 fill() 方法进行相应的填充。

【例 11-4】 Canvas API 填充图形示例, 代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title>Canvas API 填充</title>
  </head>
  <body>
    <canvas id = "mycanvas" width = "400" height = "400">
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
        .getContext("2d"); //获取渲染上下文对象
      ctx.beginPath();

```



```

    ctx.moveTo(100,100);
    ctx.lineTo(150,150);
    ctx.lineTo(50,150);
    ctx.closePath();
    ctx.stroke();
    //设置填充颜色
    ctx.fillStyle="blue";
    //填充
    ctx.fill();
  </script>
</body>
</html>

```

页面运行后的效果如图 11-7 所示。



如果路径未关闭，那么 fill() 方法会从路径结束点到开始点之间添加一条线，以关闭该路径，然后填充该路径。



图 11-7 填充示例

11.2.4 使用渐变色

在 Canvas 中绘制线条和填充图形时，可以使用渐变色。所谓渐变色是指在颜色采集上使用逐步抽样算法，让颜色逐步变化。

要使用渐变色，需要使用 CanvasGradient 对象，创建这个对象有 2 种方法：

(1) 使用线性颜色渐变方式，语法形式如下：

```
ctx.createLinearGradient(xStart,yStart,xEnd,yEnd);
```

其中，参数 xStart 和 yStart 表示渐变的起点坐标，xEnd 和 yEnd 表示渐变的终点坐标。

(2) 使用放射颜色渐变方式，语法形式如下：

```
ctx.createRadialGradient(xStart,yStart,radiusStart,
                        xEnd,yEnd,radiusEnd);
```

其中，参数 xStart 和 yStart 表示开始圆的圆心坐标，radiusStart 表示开始圆的半径，xEnd 和 yEnd 表示结束圆的圆心坐标，radiusEnd 表示结束圆的半径。

创建好 CanvasGradient 对象后，还得为其设置颜色基准，可以通过 CanvasGradient 对象的 addColorStop() 方法在渐变中某一点添加一个颜色变化。渐变中其他点的颜色会以此为基础，addColorStop 的语法形式为：

```
addColorStop(offset,color);
```

其中，参数 offset 是一个 0.0~1.0 的浮点数，表示渐变的开始点和结束点之间的一部分，为

0 对应于开始点,为 1 对应于结束点。color 指定 offset 显示的颜色,沿着渐变某一点的颜色是根据这个值以及任何其他的颜色来插值的。

创建好 CanvasGradient 对象后,将其赋给 CanvasRenderingContext2D 的 strokeStyle 或 fillStyle 属性,就可以使用渐变色绘制线条或进行填充了。

【例 11-5】 Canvas API 使用渐变色示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>Canvas API 使用渐变色</title>
  </head>
  <body>
    <canvas id="mycanvas" width="800" height="400">
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
        .getContext("2d"); //获取渲染上下文对象
      //创建线性颜色渐变方式的 CanvasGradient 对象
      var grad = ctx.createLinearGradient(60,200,160,200);
      //设置颜色基准
      grad.addColorStop(0,"yellow");
      grad.addColorStop(0.5,"green");
      grad.addColorStop(1,"red");
      ctx.beginPath();
      ctx.lineWidth=10;
      //把 CanvasGradient 对象赋给 strokeStyle 属性
      ctx.strokeStyle=grad;
      ctx.moveTo(60,200);
      ctx.lineTo(160,200);
      ctx.closePath();
      ctx.stroke();
      //创建放射颜色渐变方式的 CanvasGradient 对象
      var grad2 = ctx.createRadialGradient(300,200,0,300,200,100);
      //设置颜色基准
      grad2.addColorStop(0,"yellow");
      grad2.addColorStop(0.5,"green");
      grad2.addColorStop(1,"red");
      ctx.beginPath();
      ctx.arc(300,200,100,0,2*Math.PI);
      ctx.closePath();
      ctx.lineWidth=1;
      ctx.stroke();
      //把 CanvasGradient 对象赋给 fillStyle 属性
      ctx.fillStyle=grad2;
```

```
        ctx.fill();  
    </script>  
</body>  
</html>
```

页面运行后的效果如图 11-8 所示。

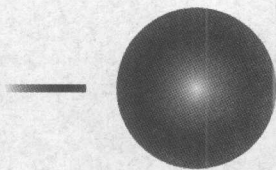


图 11-8 渐变色绘制示例

11.2.5 绘制矩形

在 Canvas 中绘制矩形的 API 有 3 个,如表 11-2 所示。

表 11-2 绘制矩形的 API

方 法	说 明
<code>rect(x,y,width,height)</code>	绘制一个左上角坐标为(x,y)、宽度为 width、高度为 height 的矩形,可以填充颜色
<code>strokeRect(x,y,width,height)</code>	绘制一个左上角坐标为(x,y)、宽度为 width、高度为 height 的矩形,不能填充颜色
<code>fillRect(x,y,width,height)</code>	绘制一个左上角坐标为(x,y)、宽度为 width、高度为 height 的“被填充”的矩形

【例 11-6】 Canvas API 绘制矩形示例,代码如下:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset = "UTF - 8">  
    <meta name = "viewport"  
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />  
    <title> Canvas API 绘制矩形</title>  
  </head>  
  <body>  
    <canvas id = "mycanvas" width = "800" height = "400">  
      你的浏览器不支持 Canvas  
    </canvas>  
    <script>  
      var ctx = document.getElementById("mycanvas").getContext("2d");  
                                              //获取渲染上下文对象  
      ctx.fillStyle = "red";
```



```
//绘制矩形,它可以被填充
ctx.rect(20,20,200,100);
ctx.stroke();
ctx.fill();
//绘制矩形,填充对它无效
ctx.strokeRect(20,150,200,100);
ctx.stroke();
ctx.fill();
//绘制填充好的矩形
ctx.fillRect(250,50,200,100);
ctx.stroke();

</script>
</body>
</html>
```

页面运行后的效果如图 11-9 所示,三种矩形的区别非常明显。

在 HTML5 中,CanvasRenderingContext2D 对象也提供了专门用于绘制圆形或弧线的方法,它的语法如下:

```
ctx.arc(x, y, radius, startAngle, endAngle, counterclockwise)
```

其中,x,y 是圆心的 x 坐标和 y 坐标,radius 是圆的半径,startAngle 和 endAngle 分别是起始弧度和终止弧度,counterclockwise 是一个布尔值。true 是逆时针方向,false 是顺时针方向(默认),如图 11-10 所示。

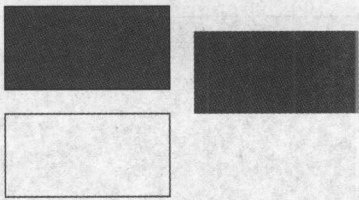


图 11-9 矩形绘制示例

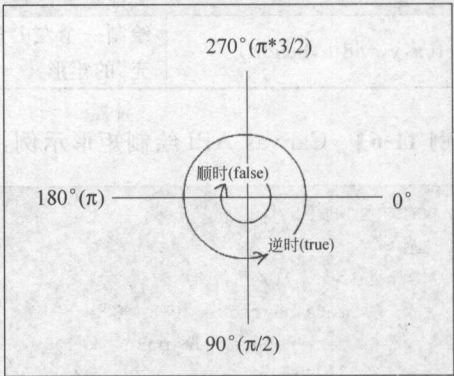


图 11-10 弧度绘制的参数示意

11.2.6 绘制圆弧

【例 11-7】 Canvas API 绘制圆弧示例,代码如下:

```
<!DOCTYPE html >
<html>
  <head>
```

```
<meta charset = "UTF-8">
<meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
<title>Canvas API 绘制弧度</title>
</head>
<body>
<canvas id = "mycanvas" width = "800" height = "400">
    你的浏览器不支持 Canvas
</canvas>
<script>
    var ctx = document.getElementById("mycanvas").getContext("2d");
    //获取渲染上下文对象

    ctx.strokeStyle = "red";
    //绘制一个圆
    ctx.arc(100,100,50,0,2 * Math.PI);
    ctx.stroke();
    ctx.moveTo(300,100);
    //顺时针绘制 0~3π/2 的圆弧
    ctx.arc(250,100,50,0,3/2 * Math.PI);
    ctx.stroke();
    ctx.moveTo(400,100);
    //逆时针绘制 0~3π/2 的圆弧
    ctx.arc(350,100,50,0,3/2 * Math.PI,true);
    ctx.stroke();
</script>
</body>
</html>
```

页面运行后的效果如图 11-11 所示。



图 11-11 弧度绘制示例

11.3 绘制文字

在 HTML5 中,还可以在 Canvas 画布上绘制我们所需的文本文字(例如游戏中的分数),其中所涉及的 CanvasRenderingContext2D 对象的主要属性和方法如下。

表 11-3 绘制文字的 API

属性和方法	说 明
font	设置绘制文字所使用的字体,该属性的用法与 CSS font 属性一致
fillText(string text, int x, int y [, int maxWidth])	从指定坐标点位置开始绘制填充的文本文字。参数 maxWidth 是可选的,如果文本内容宽度超过该参数设置,则会自动按比例缩小字体以适应宽度,对应的样式设置属性使用 fillStyle

续表

属性和方法	说 明
strokeText (string text, int x, int y[, int maxWidth])	从指定坐标点位置开始绘制非填充的文本文字(文字内部是空心的) 参数 maxWidth 是可选的,如果文本内容宽度超过该参数设置,则会 自动按比例缩小字体以适应宽度。该方法与 fillText()用法一致,不 过 strokeText()绘制的文字内部是非填充(空心)的,fillText()绘制 的文字是内部填充(实心)的。对应的样式设置属性使用 strokeStyle

【例 11-8】 Canvas API 绘制文字示例,代码如下:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title> Canvas API 绘制文字</title>
  </head>
  <body>
    <canvas id = "mycanvas" width = "800" height = "400">
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
        .getContext("2d"); //获取渲染上下文对象
      ctx.font = "30px 微软雅黑";
      //绘制实心文字
      ctx.fillStyle = "blue";
      ctx.fillText("HTML5 App 开发学习", 50, 50);
      //绘制空心文字
      ctx.strokeStyle = "red";
      ctx.strokeText("HTML5 App 开发学习", 50, 100);
    </script>
  </body>
</html>

```

页面运行后的效果如图 11-12 所示。

HTML5 App开发学习

HTML5 App开发学习

图 11-12 绘制文字示例

11.4 绘制图片

在 HTML5 中,除了在 Canvas 画布上绘制简单图形和文字,还可以在画布上绘制现有的图片文件,HTML5 游戏开发中大量应用了图片绘制技术,包括动画的实现。接下来,看看 CanvasRenderingContext2D 绘制图片所用到的 API,如表 11-4 所示。

表 11-4 绘制图片的 API

方 法	说 明
ctx. drawImage(img, x, y)	在指定的坐标点(x,y)处绘制图片对象 img
ctx. drawImage (img, x, y, width, height)	在指定的坐标点(x,y)处,以指定宽度 width 和高度 height,绘制图片对象 img
ctx. drawImage(image, sx, sy, sWidth, sHeight, x, y, width, height)	裁剪图片对象 img,起点坐标为(sx,sy),裁剪大小宽度和高度分别为(sWidth,sHeight),在指定的坐标点(x,y)处绘制图片对象 img

从表 11-4 中可以看出,3 个 drawImage 方法都用到了一个图片对象 img,在 HTML5 中,图片对象 img 的使用的基本方法如下:

```
var img = new Image();
img.onload = function(){
    //加载完成,可以进行图片绘制
}
img.src = "图片的路径(相对或绝对)";
```

下面用一个例子来演示这 3 个 drawImage 方法的使用。

【例 11-9】 Canvas API 绘制图片示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <meta name = "viewport"
content = "initial-scale=1.0,maximum-scale=1.0, user-scalable=no" />
    <title> Canvas API 绘制图片</title>
  </head>
  <body>
    <canvas id = "mycanvas" width = "800" height = "400">
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
        .getContext("2d"); //获取渲染上下文对象
      //生成图片对象
      var img = new Image();
      img.onload = function(){
```

图 11-14 绘制动画序列帧示例

```
//原版大小绘图
ctx.drawImage(this,10,10);
//指定大小绘图
ctx.drawImage(this,300,10,150,80);
//裁剪图片以指定大小绘图
ctx.drawImage(this,200,80,135,180,150,40,70,90);

    };
    img.src = "../img/baidu.png";
</script>
</body>
</html>
```

页面运行后的效果如图 11-13 所示。



图 11-13 绘制图片示例



在绘制图片时，指定的坐标实际上是图片左上角的坐标，并不是图片的中心点坐标。

11.5 擦除

众所周知，画笔一般都会与橡皮擦配套使用，以便于纠正绘画过程中的错误并重新绘画。在 HTML5 的 Canvas 中，CanvasRenderingContext2D 对象也同样给我们提供了一个可以重复使用的橡皮擦——clearRect()方法，它的语法格式为：

```
ctx.clearRect(x,y,width,height);
```

其中，x 和 y 是要清除区域的左上角坐标，width 和 height 是要擦除区域的宽度和高度。下面结合 drawImage 方法和 clearRect 方法来实现一个游戏开发中常用的动画序列帧绘制。

【例 11-10】 Canvas API 绘制动画序列帧示例，代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "UTF-8">
    <meta name = "viewport"
```

```

content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
<title> Canvas API 绘制动画帧</title>
</head>
<body>
  <canvas id = "mycanvas" width = "800" height = "400">
    你的浏览器不支持 Canvas
  </canvas>
  <script>
    var ctx = document.getElementById("mycanvas").getContext("2d");
    //获取渲染上下文对象

    //生成图片对象
    var img = new Image();
    img.onload = function(){
      //每隔 300 毫秒绘制一帧图片
      setInterval(drawBird, 300);
    };
    img.src = "../img/anim_sprite.jpg";
    var index = 0;
    var y = 0;
    function drawBird(){
      //清除 canvas 区域
      ctx.clearRect(100,100,141,85);
      //绘制动画序列的某一帧
      ctx.drawImage(img, 141 * index, y, 141, 85, 100, 100, 141, 85);
      //下一帧
      index++;
      if(index == 3&&y == 0){
        index = 0;
        y = 85;
      }else if(index == 3&&y == 85){
        index = 0;
        y = 0;
      }
    }
  </script>
</body>
</html>

```

在这个例子中,如图 11-14 所示,动画序列帧图片将一只小鸟的飞行动作逐帧分成 8 张图片,把这 8 张图片在 300 毫秒内利用 setInterval 方法进行连续绘制,并且每次绘制之前对 Canvas 画板进行一次擦除,执行后就可以看到一只小鸟飞行的动画了。

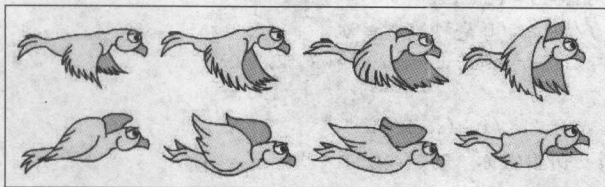


图 11-14 绘制动画序列帧示例



为提高擦除效率，在 HTML5 的游戏开发中，通常采用多层 Canvas 进行设计和局部擦除。

11.6 坐标变换

Canvas 的 API 中提供了一系列方法，可以对绘制的图形或图片进行移动、旋转、缩放和变形等，这些方法实际就是在绘制前对 Canvas 的默认坐标体系作相应的变换。如表 11-5 所示，有以下这些方法。

表 11-5 坐标变换的 API

方 法	说 明
ctx. translate(x,y)	将原点平移到指定位置(x,y)处
ctx. rotate(angle)	将坐标体系旋转 angle 弧度，正值表示顺时针方向旋转，负值表示逆时针方向旋转
ctx. scale(x,y)	将坐标体系进行缩放，x,y 是缩放因子，必须是正值
ctx. save()	将当前坐标体系状态入栈
ctx. restore()	将上一个保存的坐标体系状态从栈中再次取出，恢复该状态的所有设置

【例 11-11】 Canvas API 坐标变换示例，代码如下：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>Canvas API 坐标体系变换</title>
  </head>
  <body>
    <canvas id="mycanvas" width="400" height="400"
      style="background-color: #DBDBDB;"
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
        .getContext("2d"); //获取渲染上下文对象
      var animId; //动画刷帧 id
      //生成图片对象
      var img = new Image();
      img.onload = function(){
        //每隔 200 毫秒绘制一次
        animId = setInterval(drawRock, 200);
      };
      img.src = "../img/rock.png";
      //陨石初始坐标
```

```

var y = 50;
var x = 150;
//旋转角度增量
var rAngle = 5;
function drawRock(){
    //清屏
    ctx.clearRect(0,0,400,400);
    //保存坐标体系入栈
    ctx.save();
    //平移坐标原点到图片中心
    ctx.translate(x + img.width/2, y + img.height/2);
    //旋转坐标
    ctx.rotate(rAngle);
    //缩小坐标
    ctx.scale(0.5,0.5);
    //绘制图片
    ctx.drawImage(img, -img.width/2, -img.height/2);
    //出栈,恢复原有的坐标体系
    ctx.restore();
    y += 5;
    rAngle += 10;
    if(y <= -10){
        //停止绘制
        clearInterval(animId);
    }
}
</script>
</body>
</html>

```

页面运行后的效果如图 11-15 所示,一个陨石出现在页面上,它一边旋转一边坠落下来。这个例子比较典型,Canvas 的旋转只能旋转坐标体系,而陨石的旋转是绕自己的中心,所以必须先平移原有的坐标原点,再进行旋转,并且在对原有坐标体系更改之前,记得先保存当前状态,绘制完成后,再恢复到原始状态。

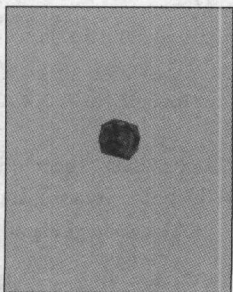


图 11-15 坠落的陨石示例

11.7 像素操作

Canvas 的强大之处还在于它提供了像素级的操作,它的 `CanvasRenderingContext2D` 对象 API 中有以下几种方法。

- `ctx.getImageData(x,y,width,height)`: 返回 `ImageData` 对象,该对象包含了画布上指定矩形的像素数据,其中 `x,y` 是所取区域的左上角坐标,`width` 和 `height` 分别是所取区域的宽度和高度。返回的 `ImageData` 对象中有个 `data` 属性,它是一个数组,数组中的每 4 个元素分别对应一个像素的 R(红色,值 0~255)、G(绿色,值 0~255)、B(蓝色,值 0~255)、A(透明度,值 0~255)。
- `ctx.putImageData(imgData,x,y)`: 将处理好的 `ImageData` 对象放回到 Canvas 画布上。
- `ctx.createImageData(width,height)`: 创建新的、空白的 `ImageData` 对象,`width` 和 `height` 是指定的宽度和高度。

【例 11-12】 Canvas API 像素操作示例,代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport">
    content="initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>Canvas API 像素操作</title>
  </head>
  <body>
    <canvas id="mycanvas" width="800" height="500">
      你的浏览器不支持 Canvas
    </canvas>
    <script>
      var ctx = document.getElementById("mycanvas")
                          .getContext("2d"); //获取渲染上下文对象

      //生成图片对象
      var img = new Image();
      img.onload = function(){
        ctx.drawImage(this,0,0,800,500);
        //获取像素数据
        var imdata = ctx.getImageData(0,0,400,250);
        //反色处理
        for(var i=0;i<imdata.data.length;i+=4){
          imdata.data[i] = 255 - imdata.data[i];
          imdata.data[i+1] = 255 - imdata.data[i+1];
          imdata.data[i+2] = 255 - imdata.data[i+2];
        }
        //放回 canvas 上
        ctx.putImageData(imdata,0,0);
      }
    </script>
  </body>
</html>
```



```

//取像素数据
var imodata = ctx.getImageData(500,250,200,200);
//透明度处理
for(var i=0;i<imodata.data.length;i+=4){
    imodata.data[i+3] *= 0.6;
}
//放回 canvas 上
ctx.putImageData(imodata,500,250);
//创建新的 imgData 对象
var imndata = ctx.createImageData(100,100);
ctx.putImageData(imndata,450,10);
};
img.src = "../img/finger.jpg";
</script>
</body>
</html>

```

页面运行后的效果如图 11-16 所示,可以看到,图片的不同区域的像素使用 Canvas 的 API 进行了反色、透明度、空白的处理。

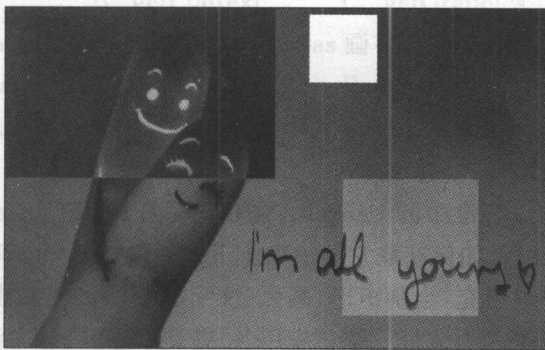


图 11-16 像素处理示例效果

11.8 实例：九宫格手势解锁

这里给出一个比较综合的 Canvas 应用实例,利用 Canvas 的各种 API 来实现一个在 App 开发中常见的“九宫格手势解锁”效果。

【例 11-13】 Canvas API 实现“九宫格手势解锁”,由于代码较多,就不在此处赘述了,请参考本书的配套源代码。

在这个例子中,先根据手机设备对 Canvas 的宽度和高度作了一定适配,然后绘制出锁的形状,并对锁的每个位置作了编号(0,1,2,...,9),如图 11-17 所示,并预先作了个固定的手势密码序列(0-4-6-7-8)。当手指在手机等设备上移动时,只要触点进入锁心的圆圈内,当前序号就算选中,并依次加入连线点数组,程序会不断擦除 Canvas,并按手指以灰色绘制连线点,最后只需要比较连线点数组序列与答案连线点是否一致,就可以作解锁成功与否的判断了。如果正确会弹出对话框提示“恭喜,正确解锁!”;如果错误,则会以红色线条将错误

连线点序列重绘一次,并提示“密码错误”,如图 11-18 所示。

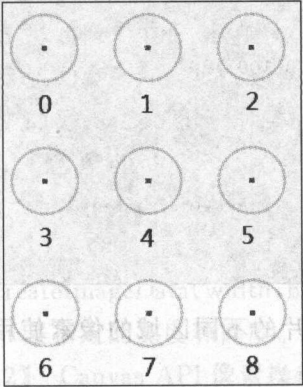


图 11-17 九宫格锁编号

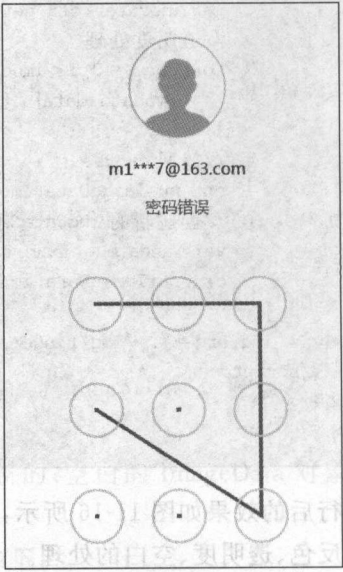


图 11-18 九宫格解锁失败效果

小结

本章介绍了 HTML5 标准中的 Canvas。先是简单介绍了 Canvas 的特点和坐标体系，< canvas >标签的使用,以及如何获取它的渲染上下文对象；接着讲解了使用相应的 API 绘制直线、贝塞尔曲线,实现填充,绘制矩形、圆弧、文字、图片,如何实现 Canvas 的擦除,以及 Canvas 的坐标变换和像素操作；最后综合应用各 API 实现了一个 App 开发中常见的“九宫格手势解锁”效果。

习题

一、选择题

- 1. 关于 Canvas 坐标体系,下面说法错误的是()。
 - A. Canvas 使用二维坐标体系,即有 x 轴和 y 轴。
 - B. 默认情况下,坐标原点位于左下角, x 轴向右为正, y 轴向上为正。
 - C. Canvas 可以对像素操作。
 - D. Canvas 可以绘制图片。
- 2. 绘制二次贝塞尔曲线的方法是()。
 - A. `quadraticCurveTo(cpx,cpy,x,y)`
 - B. `bezierCurveTo(cpx1,cpy1,cpx2,cpy2,x,y)`
 - C. `quadraticCurveTo(cpx1,cpy1,cpx2,cpy2,x,y)`
 - C. `bezierCurveTo(cpx,cpy,x,y)`

3. Canvas 的 API 中不包含以下()方法。
A. getContext() B. fill() C. controll() D. stroke()
4. Canvas 用于填充颜色设置的属性是()。
A. fillStyle B. fill C. lineWidth D. strokeStyle
5. Canvas 中为了把图片的某个部分以指定大小绘制在页面上,应该使用的方法是()。
A. drawImage(image, x, y, width, height);
B. drawImage(image, sx, sy, sWidth, sHeight, dx, dy, width, height);
C. drawImage(img, x, y);
D. 以上都不正确
6. 调用 Canvas 的 API 中的 translate 方法的作用是()。
A. 将指定的图形移动到指定位置
B. 将以后绘制的图形移动到指定的位置
C. 将 Canvas 画布的内容移动到指定的位置
D. 将 Canvas 画布的原点移动到指定的位置
7. 用于获取画布的渲染上下文对象的方法是()。
A. getContent B. getContext C. getGraphics D. getCanvas
8. 将处理好的 ImageData 对象放回到 Canvas 画布上的方法是()。
A. getImageData B. setImageData
C. putImageData D. createImageData

二、判断题

1. Canvas 的坐标原点在左下角, x 轴向右为正, y 轴向上为正。()
2. Canvas 画布的大小可以使用 CSS 定义。()
3. FillRect 可以绘制一个实心的矩形。()
4. 在 Canvas 进行坐标变换时,要使用 save 方法保存坐标状态,使用 restore 方法恢复坐标状态。()
5. 使用 Canvas 实现游戏开发,必然会用到清屏动作。()
6. Canvas API 中使用 strokeText 完成实心文字绘制。()

三、填空题

1. Canvas API 中使用_____方法绘制圆弧。
2. Canvas API 中使用_____方法或_____方法创建 CanvasGradient 渐变色对象。
3. Canvas 处理像素时,getImageData 方法返回得到的 data 数组,它的每_____个元素对应画布上的一个像素的 RGBA 值。
4. Canvas 绘制直线时,移动画笔到某个坐标点上使用的方法是_____。
5. Canvas 用于设置线条宽度的属性是_____。

四、编程题

使用 Canvas 绘图 API 实现如图 11-19 所示的饼形图效果。

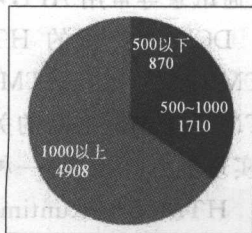


图 11-19 饼形图效果

学习目标

- 了解 HTML5+ Runtime。
- 掌握在页面中如何使用 HTML5+ API。
- 了解 HTML5+ 的模块组成。
- 掌握 HTML5+ 中的 Webview 模块。
- 了解 HTML5+ 中的 Native.js。

HTML5+ 规范对 HTML5 能力进行了有力的扩展, DCloud 公司的 HBuilder 内置了 HTML5+ Runtime, 实现了对 HTML5+ 规范的支持。本章将针对 HTML5+ Runtime 作简单介绍, 并对其中比较重要的 Webview 模块作详细的讲解。

12.1 HTML5+ Runtime 介绍

HTML5 规范对于原有的网页标准作了很多改进, 但这些功能主要是针对浏览器的。对于 HTML5 App 应用开发, 尤其是在 App 应用开发中经常会用到的一些功能, 例如二维码识别、消息推送、相机和麦克风控制、支付等功能却没有涉及。业内之前有 PhoneGap/Cordova 跨平台移动 App 开发方案, 但是它们自带 JavaScript API 太少了, 扩展 API 需要用原生语言开发, 更致命的是, 这类方案的性能不足。

HTML5+ 规范是一套 HTML5 能力的扩展规范, 定义了未存在于 HTML5 中, 但制作 App 时非常需要的扩展规范, 它隶属于 HTML5 中国产业联盟 (详见 <http://www.html5plus.org>)。封装成跨平台的 HTML5+ 规范是由 HTML5 中国产业联盟统一制定的, 它并不做厂商私有 API, 而是包括二维码、摇一摇、语音输入、地图、支付、分享、文件系统、通讯录等常用 API, 可以实现方便简单的编写和跨平台 App 应用开发。

DCloud 公司的 HTML5 Plus Runtime, 简称 HTML5+ Runtime, 完整地实现了 HTML5+ 规范。HTML5+ Runtime 是实现 HTML5+ 规范的强化浏览器引擎。它和 HTML5+ 规范之间的关系有点类似于 Chrome OS 和 W3C 的关系。HTML5+ Runtime 还实现了 Native.js, 一种通过 JavaScript 调用几十万原生 API 的技术。

HTML5+ Runtime 内置于 HBuilder, 在真机运行、打包时自动挂载。除了支持标准 HTML5 外, 还支持更多扩展的 JS API, 使得 JavaScript 的能力不输于原生。

12.2 HTML5+ 的 Demo 示例

为了演示 HTML5+ Runtime 中各 API 的使用, HBuilder 内含了一个 HTML5+ API 的 Demo 演示的源代码, 使用的方法是打开 HBuilder, 单击“文件”菜单中的“新建”命令, 选择“移动 App”后, 将弹出对话框, 如图 12-1 所示, 选择模板“Hello H5+”, 输入应用名称, 选择存放位置后, 单击“完成”按钮, 即可创建 MUI 的 Demo 演示项目。创建成功后项目的结构如图 12-2 所示, 其中的目录“doc”是每个对应 API 的说明文档页面, 目录“plus”中的每个页面都是其中一个 API 的对应 Demo 页面。当然, 我们也可以直接扫描二维码进行下载安装体验, 如图 12-3 所示。

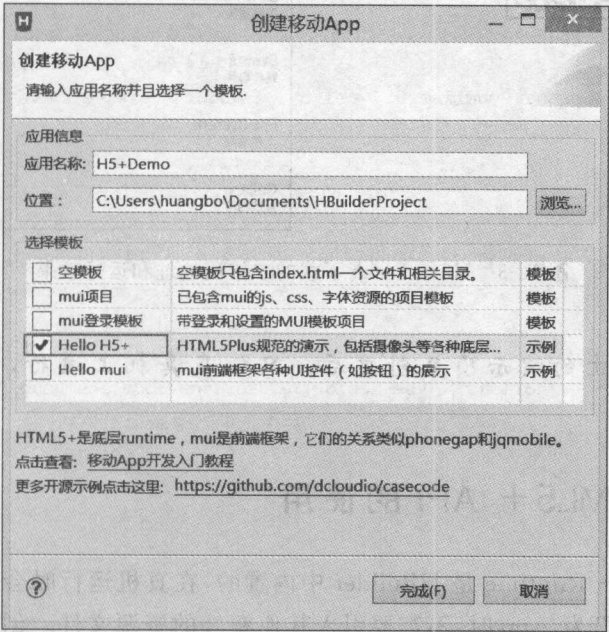


图 12-1 创建 HTML5+ API Demo 项目

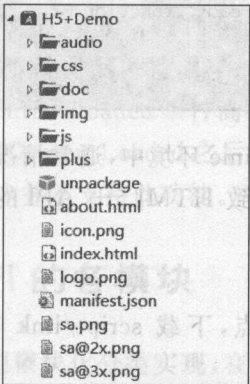


图 12-2 HTML5+ Demo 项目结构



图 12-3 H5+ Demo 下载的二维码地址和运行效果



HTML5+ 的演示项目创建后，只能在真机上运行，在浏览器中运行是没有效果的。

12.3 HTML5+ API 的使用

由于 HTML5+ Runtime 是 HBuilder 中内置的，在真机运行时会自动加载，所以在使用 HTML5+ API 开发 App 时，不需要引入其他额外的资源文件。创建页面后，通常采用以下代码调用 HTML 5+ API：

```
document.addEventListener('plusready',function () {  
    // 在这里调用 5+ API  
},false);
```

这是因为，在 HTML5+ Runtime 环境中，通常情况下需要 .html 页面解析完成后，才能让 HTML5+ API 生效，这会导致 HTML5+ API 的生效时间延后，解析中的事件执行的顺序为：

- (1) 加载页面；
- (2) 解析页面（解析 title 节点，下载 script/link 等节点引用的资源，例如 .css/.js 文件）；
- (3) 触发 DOMContentLoaded 事件；
- (4) 注入 HTML5+ API；

(5) 触发 plusready 事件。

在 HBuilder 7.5 版本之后,支持提前注入 HTML5+ API,可以在 plusready 事件触发之前调用 5+API,操作方法是在页面中添加以下节点:

```
<script src="html5plus://ready"></script>
```

【例 12-1】 HTML5+ API 提前生效处理,新建“移动 App”项目,index.html 的示例代码如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport"
content="width=device-width,initial-scale=1.0,maximum-scale=1.0,user-scalable=no" />
    <title>HTML5+ API</title>
    <script src="html5plus://ready"></script>
    <script>
      // 这里可以调用 5+ API 了,为了更好的兼容性,应使用以下代码进行判断
      if(window.plus) {
        // 在这里调用 5+ API
      } else { // 兼容老版本的 plusready 事件
        document.addEventListener('plusready', function() {
          // 在这里调用 5+ API
        }, false);
      }
    </script>
  </head>
  <body>
    Hello HTML5 plus.
  </body>
</html>
```

要注意的是,HTML 5+ API 虽然可以提前生效,但为了不引发向下兼容问题,plusready 事件并不会提前触发,该事件仍然保持原来的触发时机;另一方面,操作提前生效的 5+API 需小心,此时 DOMContentLoaded 事件高概率未触发,操作 DOM 很有可能会失败,操作 DOM 一定要等到 DOMContentLoaded 之后;另外 Android3.0 及以上平台才支持提前生效,低版本的 Android 版本并不支持此功能。

12.4 HTML5+ API 的各模块

所有 HTML5+ API 都是按照模块化分类实现,在 App 云端打包时,会根据应用中使用 HTML5+API 自动选择使用对应的模块进行打包,以优化生成安装包(apk 或 ipa)的尺寸。表 12-1 列出了 HTML5+规范中的 API 模块分类。

表 12-1 HTML5+规范中的 API 模块分类

模块名称	说明	API
Accelerometer	访问设备感应器	plus. accelerometer
Audio	访问设备麦克风	plus. audio
Cache	管理应用缓存	plus. cache
Camera	访问摄像头设备	plus. camera
Contacts	访问系统联系人信息	plus. constacts
Device	访问设备信息	plus. device, plus. display, plus. networkinfo, plus. screen
Downloader	管理文件下载任务	plus. downloader
File	访问本地文件系统	plus. io
Gallery	访问系统相册	plus. gallery
Geolocation	获取设备位置信息	plus. geolocation
Messaging	访问设备通信能力	plus. messaging
NativeObj	原生对象	plus. nativeObj
NativeUI	原生 UI 控件	plus. nativeUI
Navigator	浏览器信息	plus. navigator
Orientation	获取设备方向信息	plus. orientation
Push	管理消息推送能力	plus. push
Proximity	获取距离感应器	plus. proximity
Storage	管理应用本地数据	plus. storage
Uploader	管理文件上传任务	plus. uploader
Runtime	访问运行期环境 API	plus. Runtime
Webview	窗口管理	plus. Webview
XMLHttpRequest	跨域网络访问 API	plus. net
Zip	文件压缩与解压缩	plus. zip
Maps	地图扩展功能	plus. maps
Barcode	二维码功能	plus. barcode
Payment	支付功能	plus. payment
Share	分享功能	plus. share
Speech	语音识别功能	plus. speech
Statistic	统计功能	plus. statistic

由于 HTML5+ 各 API 的文档比较繁琐,在 12.5 节中,仅介绍最重要的一个模块——Webview 模块,其他的就不一一讲解了,各模块的示范代码请尽量参考 HTML5+ 的 Demo 项目,更详细的内容请参考官方的 API 文档。

12.5 Webview 模块

Webview 模块管理应用窗口界面,实现多窗口的逻辑控制管理操作。通过 plus.Webview 可获取应用界面管理对象。

12.5.1 Webview 的方法

App 一般都包含多个窗口,也有的是单页应用(这种应用页面可维护性很差,代码也很

复杂),各窗口在 HTML5 App 中是使用 HTML5 页面设计并实现的,但是和浏览网页不同,HTML5 App 的窗口不能依赖于<a>标签实现页面的跳转,一个窗口就是一个 Webview,可以通过控制 Webview 的切换从而控制不同界面的出现。

Webview 作为原生 App 中加载网页的组件,在 Android 和 iOS 中都有,HTML5+Runtime 中的 Webview 模块实际就是 Dcolud 公司对原生的 Webview 进行了一些封装,便于直接使用 JavaScript 进行一些调用。每个页面作为一个 Webview 对象,拥有各自的 JavaScript 运行空间,页面之间互不干扰,但是这些 Webview 对象和页面一样,它们之间也可以共享一些 storage、session 等。

过去在实现 HTML5 App 时,不可避免地会遇到一些性能问题,主要体现在动画效果不流畅,例如各界面的转场动画、下拉回弹动画、侧滑动画等。在 HTML5+ Runtime 中,完成这些效果时是调用原生 API 实现的,进而解决了 JavaScript 和 CSS 的动画性能不足的问题。

下面是 Webview 的一些重要方法。

- all: 获取应用中已创建的所有 Webview 窗口,包括所有未显示的 Webview 窗口。返回一个 WebviewObject 对象数组,按创建的先后顺序排列,即数组中第一个 WebviewObject 对象用是加载应用的入口页面。语法形式为:

```
Array[WebviewObject] plus.Webview.all();
```

- close: 关闭已经打开的 Webview 窗口,需先获取窗口对象或窗口 id,并可指定关闭窗口的动画及动画持续时间。语法形式为:

```
void plus.Webview.close(id_wvobj[,aniClose,duration,extras]);
```

id_wvobj 是窗口的 id 或窗口对象,若操作窗口对象已经关闭,则无任何效果。使用窗口 id 时,则查找对应 id 的窗口,如果有多个相同 id 的窗口,则关闭最先打开的窗口,若没有查找到对应 id 的 WebviewObject 对象,则无任何效果。

aniClose 是 String 类型,代表关闭 Webview 窗口的动画效果,取值如表 12-2 所示。

表 12-2 Webview 关闭动画效果参数

值	说 明
"auto"	自动选择动画效果,即使用显示时设置的窗口动画相对应的关闭动画
"none"	立即关闭页面,无任何动画效果
"slide-out-right"	页面从屏幕中横向向右侧滑动到屏幕外关闭
"slide-out-left"	横向向左侧滑出屏幕动画
"slide-out-top"	页面从屏幕中竖向向上侧滑动到屏幕外关闭
"slide-out-bottom"	页面从屏幕中竖向向下侧滑动到屏幕外关闭
"fade-out"	页面从不透明到透明逐渐隐藏关闭
"zoom-in"	从大逐渐缩小关闭动画
"zoom-fade-in"	从大逐渐缩小并且从不透明到透明逐渐隐藏关闭动画
"pop-out"	页面从屏幕右侧滑出消失,同时上一个页面带阴影效果从屏幕左侧滑入显示

duration 是关闭 Webview 窗口动画的持续时间,单位为 ms,如果没有设置,则使用显

示窗口动画时间。

extras 是关闭 Webview 窗口扩展参数,是个 JSON 对象,可用于指定 Webview 窗口动画是否使用图片加速。

- create: 创建 Webview 窗口,用于加载新的 HTML 页面,可通过 styles 设置 Webview 窗口的样式,创建完成后需要调用 show 方法才能将 Webview 窗口显示出来。语法形式如下:

```
WebviewObject plus.Webview.create(url[, id, styles, extras]);
```

url 是新打开 Webview 窗口要加载的 HTML 页面地址,可支持本地地址和网络地址。
id 是新窗口的标识,窗口标识可用于在其他页面中通过 getWebviewById 来查找指定的窗口,为了保持窗口标识的唯一性,应该避免使用相同的标识来创建多个 Webview 窗口。如果传入无效的字符串,则使用 url 参数作为 WebviewObject 窗口的 id 值。

extras 是创建 Webview 窗口的额外扩展参数,是一个 JSON 对象,可以用来在界面间传值。
styles 是个 JSON 对象,可以用于创建 Webview 窗口的样式(如窗口宽、高、位置等信息),它的配置属性如表 12-3 所示。

表 12-3 Webview 配置参数

属性	说 明
cachemode	String 类型,窗口的缓存模式,有 4 种取值。default 表示根据 cache-control 决定是否使用缓存数据,如果存在缓存并且没有过期,则使用本地缓存资源,否则从网络获取; cacheElseNetwork 表示只要存在缓存(即使过期)数据则使用,否则从网络获取; noCache 表示不使用缓存数据,全部从网络获取; cacheOnly 表示仅使用缓存数据,不从网络获取(注:如果没有缓存数据,则会导致加载失败)。默认使用 default
background	String 类型,窗口的背景颜色,窗口空白区域的背景模式。设置 background 为颜色值(参考 CSS Color Names,可取值/十六进制值/rgb 值/rgba 值),窗口为独占模式显示(占整个屏幕区域);设置 background 为“transparent”,则表示窗口背景透明,为非独占模式
blockNetworkImage	Boolean 类型,是否阻塞网络图片的加载。true 表示阻塞,false 表示不阻塞,默认值为 false。阻塞后 Webview 窗口将不加载页面中使用的所有网络图片,可通过 Webview 窗口对象的 setBlockNetWorkImage()方法动态修改此状态
bottom	String 类型,窗口垂直向上的偏移量,支持百分比、像素值,默认值无值(根据 top 和 height 属性值来自动计算)。当设置了 top 和 height 值时,忽略此属性值;当未设置 height 值时,可通过 top 和 bottom 属性值来确定窗口的高度
bounce	String 类型,窗口遇到边框是否有反弹效果,有 4 种取值。none 表示没有反弹效果; vertical 表示垂直方向有反弹效果; horizontal 表示水平方向有反弹效果; all 表示垂直和水平方向都有反弹效果
bounceBackground	String 类型,窗口回弹效果区域的背景,窗口回弹效果区域背景可支持颜色值或图片。颜色值格式为“#RRGGBB”,如“#FFFFFF”为设置白色背景;背景图为“url(%image path%)”,如“url(/icon.png)”为设置 icon.png 为背景图,图片采用平铺模式绘制

续表

属性	说 明
decelerationRate	Number 类型。表示窗口内容停止滑动的减速度。当 Webview 加载的内容超过其高度时,可以拖曳滑动内容,decelerationRate 属性控制手指松开后页面滑动的速度。设置值越大,手指松开后的滑动速度越快(滑动距离越长),其值域范围为 0.0~1.0,默认值为 0.989
dock	String 类型,窗口的停靠方式,当 Webview 窗口添加到另外一个窗口中时,停靠方式才会生效。有 4 种取值: top 表示控件停靠则页面顶部; bottom 表示停靠在页面底部; right 表示停靠在页面右侧; left 表示停靠在页面左侧
errorPage	String 类型,窗口加载错误时跳转的页面地址。当 Webview 窗口无法加载指定的 url 地址时(如本地页面不存在,或者无法访问的网络地址),此时会自动跳转到指定的错误页面地址(仅支持本地页面地址)。设置为“none”则关闭跳转到错误页面功能,此时页面显示 Webview 默认的错误页面内容。默认使用 5+ Runtime 内置的错误页面
hardwareAccelerated	Boolean 类型,窗口是否开启硬件加速。true 表示开启硬件加速,false 表示不开启硬件加速,默认情况下 5+ Runtime 会根据设备实际支持情况自动选择是否开启硬件加速,可以通过plus.Webview.defaultHardwareAccelerated()方法获取默认 Webview 是否开启硬件加速。由于不同设备对硬件加速的支持情况存在差异,开启硬件加速能加速 HTML 页面的渲染,但也会消耗更多的系统资源,从而导致在部分设备上可能出现闪屏、发虚、分块渲染等问题,因此,如果在特定设备的特定页面出现以上问题,需要手动设置关闭硬件加速来避免
height	String 类型,窗口的高度,支持百分比、像素值,默认为 100%。当未设置 height 属性值时,优先通过 top 和 bottom 属性值来计算窗口的高度
kernel	String 类型,窗口使用的内核,有两种取值。WKWebview 表示在 iOS8.0 及以上系统使用 WKWebview 内核,低版本下仍然使用 UIWebview 内核; UIWebview 表示在所有版本上都使用 UIWebview 内核。默认值为“UIWebview”。使用 UIWebview 内核会有更好的性能,但在功能上有些限制,目前已知的问题有: ①不支持设置 cookie,即 plus.navigator.setCookie() API 无法使用; ②本地的 HTML 页面中的 XHR 不支持跨域访问,需使用 plus.net.XMLHttpRequest 来替换; ③不支持使用 WebSQL,需使用 IndexedDB 来替换; ④不支持 js 原生混淆功能,需使用前端 js 混淆来替换
left	String 类型,窗口水平向右的偏移量,支持百分比、像素值,默认值为 0。未设置 left 属性值时,优先通过 right 和 width 属性值来计算窗口的 left 位置
margin	String 类型,窗口的边距,用于定位窗口的位置,支持 auto。auto 表示居中; 若设置了 left、right、top、bottom 则对应的边距值失效
mask	String 类型,窗口的遮罩,用于设置 Webview 窗口的遮罩层样式。遮罩层会覆盖 Webview 中所有内容,包括子 Webview,并且截获 Webview 的所有触屏事件,此时 Webview 窗口的单击操作会触发 maskClick 事件。字符串类型,可取值: rgba+格式字符串用于定义纯色遮罩层样式,如“rgba(0,0,0,0.5)”,表示黑色半透明; none 表示不使用遮罩层; 默认值为 none,即无遮罩层
opacity	Number 类型,窗口的不透明度。0 为全透明,1 为不透明,默认值为 1

续表

属性	说 明
popGesture	String 类型,窗口的侧滑返回功能有 3 种取值。none 表示无侧滑返回功能;close 表示侧滑返回关闭 Webview 窗口;hide 表示侧滑返回隐藏 Webview 窗口。仅 iOS 平台支持
render	String 类型,窗口渲染模式。支持以下属性值:onscreen 表示 Webview 窗口在屏幕区可见时渲染,不可见时不进行渲染,此时能减少内存使用量;always 表示 Webview 在任何时候都渲染,在内存较大的设备上使用,被遮挡的窗口在此中模式下显示的时候会有更流畅的效果。默认值为 onscreen。仅 Android 平台支持
right	String 类型,窗口水平向左的偏移量,支持百分比、像素值,默认无值(根据 left 和 width 属性值来自动计算)。当设置了 left 和 width 值时,忽略此属性值;当未设置 width 值时,可通过 left 和 bottom 属性值来确定窗口的宽度
scalable	Boolean 类型,窗口是否可缩放。设置为 true 时,用户可通过双指操作放大或缩小页面,此时 html 页面可通过 meta 节点来限制页面不可缩放。窗口设置为 false 时,用户不可通过双指操作放大或缩小页面,即使页面中的 meta 节点也无法开启可缩放功能。默认值为 false,即不可缩放
scrollIndicator	String 类型,用于控制窗口滚动条样式,有 4 种取值。all 表示垂直和水平滚动条都显示;vertical 表示仅显示垂直滚动条;horizontal 表示仅显示水平滚动条;none 表示垂直和水平滚动条都不显示。默认值为 all,即垂直和水平滚动条都显示。注意:显示滚动条的前提条件是窗口中的内容超过窗口显示的宽或高
scrollsToTop	Boolean 类型,单击设备的状态栏时是否滚动返回至顶部。true 表示单击设备的状态栏可以滚动返回至顶部,false 表示单击设备的状态栏不可以,默认值为 true。此功能仅支持 iOS 平台,在 iPhone 上有且只有一个 Webview 窗口的 scrollsToTop 属性值为 true 时才生效,所以在显示和关闭 Webview 窗口时,需动态更新所有 Webview 的 scrollsToTop 值,以确保此功能生效
softinputMode	String 类型,弹出系统软键盘模式,有两种选值。adjustPan 表示弹出软键盘时 Webview 窗口自动上移,以保证当前输入框可见;adjustResize 表示自动调整 Webview 窗口大小(屏幕区域减去软键盘区域),同时自动滚动 Webview 保证输入框可见。默认值为“adjustPan”
top	String 类型,窗口垂直向下的偏移量,支持百分比、像素值,默认值为 0px。未设置 top 属性值时,优先通过 bottom 和 height 属性值来计算窗口的 top 位置
position	String 类型,Webview 窗口的排版位置。当 Webview 窗口添加到另外一个窗口中时,排版位置才会生效,排版位置决定子窗口在父窗口中的定位方式,有 3 种取值:static 表示控件在页面中正常定位,如果页面存在滚动条则随窗口内容滚动;absolute 表示控件在页面中绝对定位,如果页面存在滚动条不随窗口内容滚动;dock 表示控件在页面中停靠,停靠的位置由 dock 属性值决定。默认值为 absolute
width	String 类型,窗口的宽度,支持百分比、像素值,默认为 100%。未设置 width 属性值时,可同时设置 left 和 right 属性值改变窗口的默认宽度
zindex	Number 类型,窗口的堆叠顺序值。拥有更高堆叠顺序的窗口总是会处于堆叠顺序较低的窗口的前面,拥有相同堆叠顺序的窗口后调用 show 方法则在前面

- currentWebview: 获取当前窗口的 WebviewObject 对象。语法形式如下:


```
WebViewObject plus.Webview.currentWebView();
```

- `getWebViewById`: 在已创建的窗口列表中查找指定标识的 `WebView` 窗口并返回。若没有查找到指定标识的窗口则返回 `null`, 若存在多个相同标识的 `WebView` 窗口, 则返回第一个创建的 `WebView` 窗口。语法形式如下:

```
WebViewObject plus.Webview.getWebViewById(id);
```

`id` 是 `String` 类型, 要查找的 `WebView` 窗口标识。

- `getLaunchWebView`: 获取应用首页 `WebViewObject` 窗口对象。语法形式如下:

```
WebViewObject plus.Webview.getLaunchWebView();
```

- `getTopWebView`: 获取应用显示栈顶的 `WebViewObject` 窗口对象。语法形式如下:

```
WebViewObject plus.Webview.getTopWebView();
```

- `hide`: 根据指定的 `WebViewObject` 对象或 `id` 隐藏 `WebView` 窗口, 使得窗口不可见。语法形式如下:

```
void plus.Webview.hide(id_wvobj[, aniHide, duration, extras]);
```

`id_wvobj` 是窗口的 `id` 或窗口对象, 若窗口对象已经隐藏, 则无任何效果。使用窗口 `id` 时, 则查找对应 `id` 的窗口, 如果有多个相同 `id` 的窗口, 则操作最先打开的那个窗口, 若没有查找到对应 `id` 的 `WebViewObject` 对象, 则无任何效果。

`aniHide` 是 `String` 类型, 隐藏 `WebView` 窗口的动画效果, 与关闭 `WebView` 时取值类似。

`duration` 是指隐藏 `WebView` 窗口动画的持续时间单位为 `ms`, 如果没有设置则使用默认窗口动画时间。

`extras` 是一个 `JOSN` 对象, 隐藏 `WebView` 窗口扩展参数, 可用于指定 `WebView` 窗口动画是否使用图片加速。

- `open`: 创建并显示 `WebView` 窗口, 用于加载新的 `HTML` 页面, 可通过 `styles` 设置 `WebView` 窗口的样式, 创建完成后自动将 `WebView` 窗口显示出来。语法形式如下:

```
WebViewObject plus.Webview.open(url[, id, styles,  
                                aniShow, duration, showedCB]);
```

`url` 是新打开 `WebView` 窗口要加载的 `HTML` 页面地址, 可支持本地地址和网络地址。

`id` 是 `String` 类型, 打开窗口的标识。

`styles` 是一个 `JSON` 配置对象, 用于创建 `WebView` 窗口的样式 (如窗口宽、高、位置等信息)。

anishow 是个 String,用于显示 Webview 窗口的动画效果,取值如表 12-4 所示。

表 12-4 Webview 打开动画效果参数

值	说 明
auto	自动选择动画效果,使用上次显示窗口设置的动画效果,如果是第一次显示则默认动画效果“none”
none	立即显示页面,无任何动画效果,页面显示默认的动画效果
slide-in-right	页面从屏幕右侧外向内横向滑动显示
slide-in-left	页面从屏幕左侧向右横向滑动显示
slide-in-top	页面从屏幕上侧向下竖向滑动显示
slide-in-bottom	页面从屏幕下侧向上竖向滑动显示
fade-in	页面从完全透明到不透明逐渐显示
zoom-out	页面在屏幕中间从小到大逐渐放大显示
zoom-fade-out	页面在屏幕中间从小到大逐渐放大并且从透明到不透明逐渐显示
pop-in	页面从屏幕右侧滑入显示,同时上一个页面带阴影效果从屏幕左侧滑出隐藏

duraion 是显示 Webview 窗口动画的持续时间,单位为 ms,如果没有设置,则使用默认窗口动画时间 600ms。

showedCB 是 Webview 窗口显示完成的回调函数,当指定 Webview 窗口显示动画执行完毕时,触发回调函数,窗口无动画效果时,也会触发此回调。

- show: 显示已创建或隐藏的 Webview 窗口,需先获取窗口对象或窗口 id,并可指定显示窗口的动画及动画持续时间。语法形式如下:

```
void plus.Webview.show(id_wvobj[,aniShow,duration,showedCB,extras]);
```

id_wvobj 是要显示 Webview 窗口 id 或窗口对象。若操作 Webview 窗口对象显示,则无任何效果。使用窗口 id 时,则查找对应 id 的窗口,如果有多个相同 id 的窗口,则操作最先创建的窗口,若没有查找到对应 id 的 WebviewObject 对象,则无任何效果。

aniShow 参数与 open 方法中的一致。

duration 是显示 Webview 窗口动画的持续时间,单位为 ms,如果没有设置,则使用默认窗口动画时间 600ms。

showedCB 是 Webview 窗口显示完成的回调函数。当指定 Webview 窗口显示动画执行完毕时触发回调函数,窗口无动画效果(如"none"动画效果)时,也会触发此回调。

extras 参数与 hide()方法中的一致。

- defaultHardwareAccelerated: 获取 Webview 默认是否开启硬件加速。由于不同设备对于硬件加速的支持情况存在差异,开启硬件加速能加速 HTML 页面的渲染,但也会消耗更多的系统资源,从而导致在部分设备上可能出现闪屏、发虚、分块渲染等问题,因此 HTML5+ Runtime 会根据设备实际支持情况自动选择是否开启硬件加速。关闭硬件加速则可能会导致 Webview 页面无法支持< video >标签播放视频等问题,如果在特定情况下需要调整修改默认开启硬件加速的行为,则可通过该方法获取当前设备默认是否开启硬件加速状态。语法形式如下:

```
Boolean plus.Webview.defaultHardwareAccelerated();
```

12.5.2 WebviewObject

从 12.5.1 节 Webview 模块的很多方法可以看出,它返回的对象都是一个 WebviewObject,这是 Webview 窗口对象,用于操作加载 HTML 页面的窗口。下面介绍它的 id 属性和一些重要的方法。

1. id 属性

Webview 窗口的标识,是一个 String 类型,在打开或创建 Webview 窗口时设置,如果没有设置窗口标识,此属性值为 undefined。所以在创建或打开 Webview 窗口时,应该尽量为其设置 id 属性,并保证它的值的唯一性。如果要获取应用入口页面所属的 Webview 窗口的 id 属性,其标识为应用的 %AppID%,可通过 plus.Runtime.Appid 获取(如果是在 HBuilder 真机运行获取的是固定值“HBuilder”,需要提交 App 云端打包后运行才能获取真实的 AppID 值)。



由于页面的名字是唯一的,HTML5+ Runtime 中推荐使用页面名作为 id 属性值。

2. 方法

- addEventListener: 向 Webview 窗口添加事件监听器,当指定的事件发生时,将触发监听函数的执行。可多次调用此方法向 Webview 添加多个监听器,当监听的事件发生时,将按照添加的先后顺序执行。语法形式为:

```
wobj.addEventListener(event,listener[,capture]);
```

event 是一个 String 类型,代表 Webview 窗口的事件类型,相应说明见表 12-5。

表 12-5 Webview 窗口的事件类型

值	说 明
close	窗口关闭事件
dragBounce	窗口顶部下拉回弹事件
slideBounce	窗口左右侧侧滑回弹事件
error	窗口加载错误事件
hide	窗口隐藏事件
loading	窗口页面开始加载事件
loaded	窗口页面加载完成事件
maskClick	窗口显示遮罩层时单击事件
show	窗口显示事件
popGesture	窗口侧滑返回事件
titleUpdate	加载页面标题更新事件

listener 是监听事件发生时执行的回调函数,其输入参数是一个事件参数 Event 对象,它的 target 保存触发此事件的 Webview 窗口对象。

capture 是布尔值,代表捕获事件流顺序。

- Append: 将另一个 Webview 窗口作为子窗口添加到当前 Webview 窗口中,添加后其所有权归父 Webview 窗口,当父窗口关闭时,子窗口自动关闭。语法形式为:

```
void wobj.Append(Webview );
```

Webview 是被添加的子 Webview 窗口或 View 控件对象,被添加的 Webview 窗口需通过 plus.Webview.create 方法创建,并且不能调用其 show 方法进行显示。父窗口显示时,子窗口会自动显示,父窗口隐藏时,子窗口也会自动隐藏,被添加的 View 控件需通过 new plus.nativeObj.View() 创建,添加到 Webview 窗口后所有权一起转移(即 Webview 关闭后 View 控件也自动关闭)。

- AppendJsFile: 添加 Webview 窗口预加载 js 文件,对于一些网络 HTML 页面,在无法修改 HTML 页面时可通过此方法自动加载本地 js 文件。当 Webview 窗口跳转到新页面时,也会自动加载指定的 js 执行,添加多个 js 文件将按照添加的先后顺序执行。语法形式为:

```
wobj.AppendJsFile(file);
```

file 是 String 类型,窗口预加载的 js 文件地址。js 文件路径只支持本地文件,应该使用扩展相对路径类型的文件,如 _www/preload.js。

- back: Webview 窗口历史记录操作,后退到窗口上次加载的 HTML 页面。如果窗口历史记录中没有可后退的页面,则不触发任何操作。语法形式为:

```
void wobj.back();
```

- children: 获取添加到 Webview 窗口中的所有子 Webview 窗口,如果没有子 Webview 窗口则返回空数组。语法形式为:

```
Array[WebviewObject] wobj.children();
```

- clear: 清除原生窗口的内容,用于重置原生窗口加载的内容,清除其加载的历史记录等内容。语法形式为:

```
void wobj.clear();
```

- close: 关闭并销毁 Webview 窗口,可设置关闭动画和动画持续时间。语法形式为:

```
void wobj.close([aniClose,duration,extras]);
```

aniClose 是关闭 Webview 窗口动画效果,如果没有指定关闭窗口动画,则使用默认值“auto”,即使用显示时设置的窗口动画相对应的关闭动画。这和 Webview 模块的 close 方法是一致的。

duration 是关闭 Webview 窗口的动画持续时间,单位为 ms,默认为窗口 show 方法设置的动画时间。

extras 是关闭 Webview 窗口扩展参数,用于指定 Webview 窗口动画是否使用图片加速。

- draw: 将 Webview 窗口的可视区域截屏并绘制到 Bitmap 图片对象中。语法形式为:

```
void wobj.draw(bitmap, successCallback, errorCallback);
```

bitmap 是要绘制的图片对象,是一个 plus.nativeObj.Bitmap 对象,如果图片中已经存在内容则覆盖,如果截屏绘制失败,则保留之前的图片内容。

successCallback 是截屏绘制操作成功回调。

errorCallback 是截屏绘制操作失败回调,输入参数会返回失败信息。

- evalJS: 在 Webview 窗口中执行 JS 脚本。语法形式为:

```
void wobj.evalJS(js);
```

js 是要在窗口中运行的脚本字符串。

- forward: Webview 窗口历史记录操作,前进到窗口上次加载的 HTML 页面。如果窗口历史记录中没有可前进的页面,则不触发任何操作。语法形式为:

```
void wobj.forward();
```

- getURL: 获取 Webview 窗口加载 HTML 页面的地址。语法形式为:

```
String wobj.getURL();
```

- hide: 隐藏 Webview 窗口可保存已加载 HTML 页面的上下文数据,能降低应用使用的系统资源,通过 show 方法可将隐藏的 Webview 窗口显示出来。语法形式为:

```
void wobj.hide(aniHide, duration, extras);
```

aniHide 是隐藏 Webview 窗口动画效果,如果没有指定隐藏窗口动画,则使用默认动画效果“none”。动画效果的值与 close 方法一致。

duration 是隐藏 Webview 窗口的动画持续时间,单位为 ms,默认为窗口 show 方法设置的动画时间。

extras 是隐藏 Webview 窗口扩展参数,用于指定 Webview 窗口动画是否使用图片加速。

- `isHardwareAccelerated`: 查询 Webview 窗口是否开启硬件加速。语法形式为:

```
Boolean wobj.isHardwareAccelerated();
```

- `loadURL`: 从新的 URL 地址加载页面, 如果 url 地址无效, 将导致页面显示失败。语法形式为:

```
void wobj.loadURL(url);
```

url 是要加载的页面 URL 地址。

- `nativeInstanceObject`: 获取 Webview 窗口对象的原生 (Native. JS) 实例对象, Android 平台返回 Webview 窗口对象的 `android.webkit.Webview` 实例对象, iOS 平台返回 Webview 窗口对象的 `UIWebView` 实例对象。语法形式为:

```
InstanceObject wobj.nativeInstanceObject();
```

- `opener`: 获取当前 Webview 窗口的创建者, 创建者为调用 `plus.Webview.open` 或 `create` 方法创建当前窗口的 Webview 窗口。语法形式为:

```
WebviewObject wobj.opener();
```

- `parent`: 获取当前 Webview 窗口的父窗口, Webview 窗口作为子窗口添加到其他 Webview 窗口中时有效。语法形式为:

```
WebviewObject wobj.parent();
```

- `resetBounce`: 开启窗口回弹效果后, 当窗口中展现的内容滚动到头(顶部或底部), 再拖曳时, 窗口整体内容将跟随移动, 松开后自动回弹到停靠位置。这时需要调用此方法来重置窗口的回弹位置, 窗口将采用动画方式回弹到其初始显示的位置。语法形式为:

```
void wobj.resetBounce();
```

- `remove`: 从当前 Webview 窗口移除指定的子 Webview 窗口, 若指定的 Webview 对象不是当前窗口的子窗口, 则无任何作用。移除后, 子 Webview 窗口不会关闭, 需要调用其 `close` 方法才能真正关闭并销毁。语法形式为:

```
void wobj.remove(Webview);
```

Webview 是要移除的 Webview 窗口。

- `setBlockNetworkImage`: 设置 Webview 窗口是否阻塞加载页面中使用的网络图片。

语法形式为：

```
void wobj.setBlockNetworkImage(block);
```

block 是布尔值,true 表示不加载页面中使用的网络图片,false 表示加载页面中使用的网络图片。

- setPullToRefresh: 设置 Webview 窗口的下拉刷新效果。开启 Webview 窗口的下拉刷新功能,显示窗口内置的下拉刷新控件样式。语法形式为:

```
void wobj.setPullToRefresh(style,refreshCB);
```

style 是 Webview 窗口下拉刷新样式参数,可设置窗口内置的下拉刷新控件在各种状态显示的文字等内容,这是一个 JSON 配置对象,配置的参数如表 12-6 所示。

表 12-6 style 下拉刷新配置属性

属 性	说 明
support	Boolean 类型,是否开启 Webview 窗口的下拉刷新功能
height	String 类型,窗口的下拉刷新控件高度,支持百分比(如“10%”)和像素值(如“50px”)
range	String 类型,窗口的可下拉拖曳的范围,支持百分比(如“10%”)和像素值(如“50px”)
contentdown	JSON 类型,在下拉可刷新状态时显示的内容,支持 caption 属性:在下拉可刷新状态时,下拉刷新控件上显示的标题内容
contentover	JSON 类型,在释放可刷新状态时显示的内容,支持 caption 属性:在释放可刷新状态时下拉刷新控件上显示的标题内容
contentrefresh	在正在刷新状态时显示的内容,支持 caption 属性:在正在刷新状态时下拉刷新控件上显示的标题内容

refreshCB 是 Webview 窗口下拉刷新事件回调,用户操作窗口的下拉刷新触发窗口刷新事件时触发。

- setStyle: 设置 Webview 窗口的样式,如窗口位置、大小、背景色等。语法形式为:

```
void wobj.setStyle(styles);
```

styles 是要设置的窗口样式,这是一个 JSON 配置对象,配置的参数和 plus.Webview.open 方法中的一致。

- getStyle: 获取 Webview 窗口的样式,返回一个 JSON 对象,和 setStyle 的参数一致,语法形式为:

```
WebviewStyles wobj.getStyle();
```

- show: 当调用 plus.Webview.create 方法创建 Webview 窗口后,需要调用其 show 方法才能显示,并可设置窗口显示动画及动画时间。Webview 窗口被隐藏后,也可调用此方法来重新显示。语法形式为:

```
void wobj.show([aniShow,duration,showedCB,extras]);
```

aniShow 是 Webview 窗口显示动画类型如果没有指定窗口动画类型,则使用默认值“none”,即无动画。

duration 是 Webview 窗口显示动画持续时间,单位为 ms,默认使用动画类型希望对应的默认时间。

showedCB 是 Webview 窗口显示完成的回调函数,当指定 Webview 窗口显示动画执行完毕时触发回调函数,窗口无动画效果(如“none”动画效果)时也会触发此回调。

extras 是显示 Webview 窗口扩展参数,可用于指定 Webview 窗口动画是否使用图片加速。

12.5.3 常见的一些 UI 效果

12.5.2 节介绍了 Webview 和 WebviewObject 的一些方法和属性,灵活地掌握它们有助于制作一些意想不到的界面 UI 效果,在这里将示范如何使用它们。

【例 12-2】 Webview 窗口的下拉刷新效果,新建“移动 App”项目,选择“空模板”,其中的 index.html 的示例代码如下:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport"
      content="initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
    <meta name="HandheldFriendly" content="true" />
    <meta name="MobileOptimized" content="320" />
    <title>下拉刷新示例</title>
    <style type="text/css">
      li {
        padding: 1em;
        border-bottom: 1px solid #eaeaea;
      }
      li:active {
        background: #f4f4f4;
      }
    </style>
  </head>
  <body>
    <ul id="list" style="list-style:none;margin:0;padding:0;">
      <li><span>Item 1</span></li>
      <li><span>Item 2</span></li>
      <li><span>Item 3</span></li>
      <li><span>Item 4</span></li>
      <li><span>Item 5</span></li>
      <li><span>Item 6</span></li>
```



```

<li><span>Item 7</span></li>
<li><span>Item 8</span></li>
<li><span>Item 9</span></li>
<li><span>Item 10</span></li>
</ul>
<script>
    var ws = null;
    var list = null;
    document.addEventListener('plusready', function() {
        ws = plus.Webview.currentWebview();
        //配置下拉刷新
        ws.setPullToRefresh({
            support: true,
            height: "50px",
            range: "200px",
            contentdown: {
                caption: "下拉可以刷新"
            },
            contentover: {
                caption: "释放立即刷新"
            },
            contentrefresh: {
                caption: "正在刷新..."
            }
        }, onRefresh);
        plus.nativeUI.toast("下拉可以刷新");
    });
    // DOM 构建完成获取列表元素
    document.addEventListener("DOMContentLoaded", function() {
        list = document.getElementById("list");
    })
    // 刷新页面
    var count = 10;
    function onRefresh() {
        setTimeout(function() {
            if(list) {
                var item = document.createElement("li");
                item.innerHTML = "<span>Item " +
                    (++count) + "</span>";
                list.insertBefore(item, list.firstChild);
            }
            //结束刷新动作
            ws.endPullToRefresh();
        }, 2000);
    }
</script>
</body>
</html>

```


程序在真机上的运行效果如图 12-4 所示,当在手机显示的程序窗口中用手指向下滑动时,会自动出现常见的下拉刷新新效果。

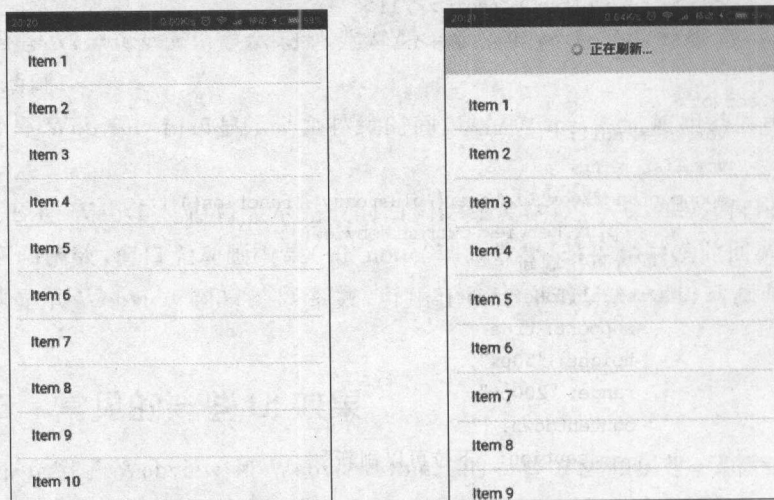


图 12-4 下拉刷新真机运行效果

【例 12-3】 Webview 窗口的跳转和回退功能,新建“移动 App”项目,创建好 3 张 HTML5 页面: index.html、second.html、third.html,它们的示例代码如下:

(1) index.html 的示例代码:

```
<!DOCTYPE html >
<html>
  <head>
    <meta charset = "utf - 8">
    <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
    <title></title>
  </head>
  <body>
    <h1>界面 1 </h1>
    <input type = "button" value = "打开界面 2" id = "btnOpen" />
    <script type = "text/javascript">
      document.addEventListener('plusready', function() {
        // Android 处理返回键
        plus.key.addEventListener('backbutton', function() {
          //退出程序
          plus.Runtime.quit();
        }, false);
        document.getElementById("btnOpen").onclick = function() {
          //查找 second.html 的 Webview
          var second =
            plus.Webview.getWebviewById("second.html");
          if(second) {
            //已存在则直接显示
```

```

        second.show();
    } else {
        //不存在就创建并显示
        plus.Webview.open("second.html", "second.html");
    }
}
});
</script>
</body>
</html>

```

(2) second.html 的示例代码:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf - 8">
        <meta name = "viewport"
content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
        <title></title>
    </head>
    <body>
        <h1>界面 2</h1>
        <input type = "button" value = "回到界面 1" id = "btnBack" />
        <input type = "button" value = "打开界面 3" id = "btnOpen" />
        <script type = "text/javascript">
            document.addEventListener('plusready', function() {
                // Android 处理返回键
                plus.key.addEventListener('backbutton', function() {
                    GoBackToFist();
                }, false);
                document.getElementById("btnOpen").onclick = function() {
                    //查找 third.html 的 Webview
                    var third = plus.Webview.getWebviewById("third.html");
                    if(third) {
                        //已存在则直接显示
                        third.show();
                    } else {
                        //不存在就创建并显示
                        plus.Webview.open("third.html", "third.html");
                    }
                };
                document.getElementById("btnBack").onclick = function() {
                    GoBackToFist();
                };
                //返回到上一个窗口
                function GoBackToFist(){
                    var Webview1 = plus.Webview.currentWebview().opener();

```



```

        Webview1.show();
    }
    });
</script>
</body>
</html>

```

(3) third.html 的示例代码:

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1.0,
maximum-scale=1.0, user-scalable=no" />
    <title></title>
  </head>
  <body>
    <h1>界面 3</h1>
    <input type="button" value="回到界面 2" id="btnBack2"/>
    <input type="button" value="回到界面 1" id="btnBack1"/>
    <script>
      document.addEventListener('plusready', function() {
        plus.key.addEventListener('backbutton', function() {
          GoBackToSecond();
        }, false);
        //显示当前的 Webview 数目和 url 地址
        var Webviews = plus.Webview.all();
        console.log("当前一共有" + Webviews.length + "个窗口\r\n");
        for(var i = 0; i < Webviews.length; i++){
          console.log(Webviews[i].getURL());
        }
        document.getElementById("btnBack1").onclick = function(){
          var Webview1 = plus.Webview.getLaunchWebview();
          Webview1.show();
        };
        document.getElementById("btnBack2").onclick = function(){
          GoBackToSecond();
        };
        function GoBackToSecond(){
          var Webview2 = plus.Webview.currentWebview().opener();
          Webview2.show();
        }
      });
    </script>
  </body>
</html>

```


这个项目在真机上运行后的效果,以及窗口之间的跳转关系如图 12-5 所示。从这个例子可以看出,使用 open 方法创建窗口后会立即将其显示,而 create 方法只是创建并不显示,在创建新窗口时最好给予 id 属性,这样便于在后面轻松找到该窗口,对于启动后的首页一般采用 getLaunchWebview 找到。另外,在创建窗口时,窗口要避免重复创建,不需要时可以进行隐藏。

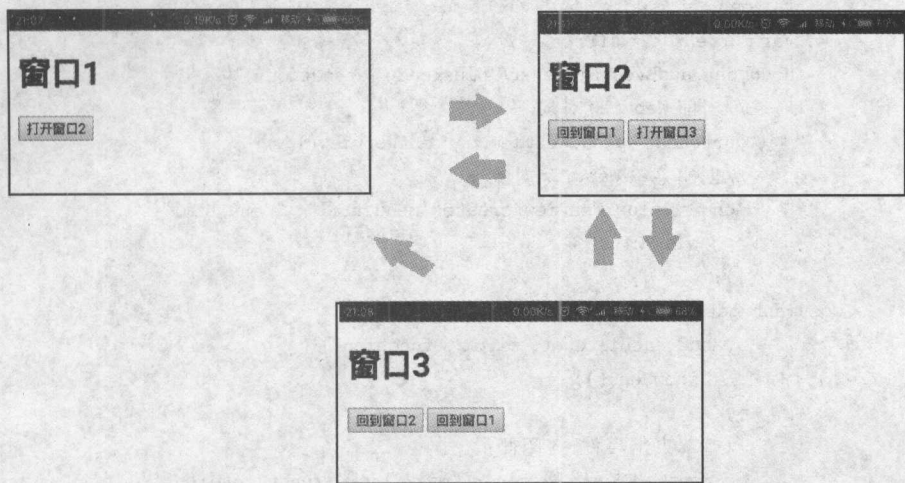


图 12-5 Webview 窗口跳转效果

【例 12-4】 Webview 侧滑菜单效果。

(1) 打开课件所附带的练习项目,分别在 Chrome 浏览器中用“移动设备模式”查看“index.html”的页面效果和“menu.html”的页面效果,如图 12-6 所示。

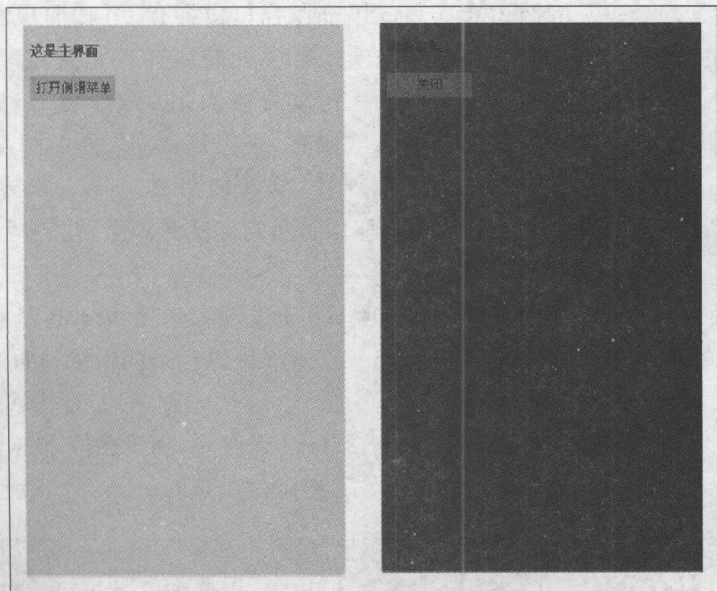


图 12-6 页面原始效果

例(2) 在 index.html 页面中修改代码为:

```
<body>
  <h4>这是主界面</h4>
  <input type="button" value="打开侧滑菜单" id="btnOpen" />
  <script>
    var menu = null;
    var current = null;
    document.addEventListener('plusready', function() {
      //当前 Webview 对象
      current = plus.Webview.currentWebview();
      //创建菜单 Webview 对象
      menu = plus.Webview.create("menu.html", "menu.html", {
        width: "40%"
      });
    });
    document.getElementById('btnOpen')
      .addEventListener('click', function() {
        showMenu();
      });
    //遮罩点击事件,关闭侧滑
    current.addEventListener("maskClick",function(e){
      closeMenu();
    });
    //显示菜单
    function showMenu(){
      current.setStyle({
        mask:"rgba(0,0,0,0.5)",
        left: "40%"
      });
      menu.setStyle({
        left:"0"
      });
      menu.show("slide-in-left",400);
    }
    //关闭菜单
    function closeMenu(){
      current.setStyle({
        left: "0",
        mask:"none"
      });
      menu.setStyle({
        left:"- 40%"
      });
    }
  </script>
</body>
```

(3) 在 menu.html 页面中加入以下代码:

```
<body>
  <h4>侧滑菜单</h4>
  <input type="button" value="关闭" id="btnClose"/>
  <script type="text/javascript">
    document.addEventListener('plusready', function() {
      var index = plus.Webview.currentWebview().opener();
      var current = plus.Webview.currentWebview();
      document.getElementById('btnClose')
        .addEventListener('click',function () {
          //关闭 index 的遮罩,并恢复原位
          index.setStyle({
            left: "0",
            mask:"none"
          });
          //向左移动当前 webview
          current.setStyle({
            left:"- 40 % "
          });
        });
    });
  </script>
</body>
```

(4) 在真机上运行后,在主界面上单击“打开侧滑菜单”,出现如图 12-7 所示的侧滑菜单效果,在菜单上单击“关闭”按钮,将关闭侧滑菜单效果。

侧滑菜单的原理如下:在主窗口启动时用 create 方法创建了一个菜单 Webview,它并不显示,这个窗口的宽度只有屏幕的 40%,当需要显示侧滑菜单时,先在主界面 Webview 中创建了一个遮罩层显示,并将主窗口在屏幕上的位置向右移动了 40%,再将菜单 Webview 也以动画的方式向右滑进来,直到左边缘与屏幕对齐。关闭侧滑菜单时,主窗口 Webview 和菜单 Webview 都反向移动,直到回到原位,并关闭遮罩层。这里可以看出 Webview 生成后它的位置以及宽度都是可以通过 HTML5+ API 中的 Webview 相应的方法进行控制的。

【例 12-5】 Webview 窗口底部 Tab bar 切换窗口示例。

(1) 打开课件所附带的练习项目,在 Chrome 浏览器中以“移动设备模式”查看 index.html 的页面效果,如图 12-8 所示。

(2) 创建 4 张不同的页面,分别为 first.html、second.html、third.html、fourth.html,内容基本相同,下面是 first.html 页面的示例:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
```



```

<title></title>
<meta name="viewport" content="initial-scale=1.0,
maximum-scale=1.0, user-scalable=no" />
</head>
<body background:lavender;>
    First 窗口
</body>
</html>

```

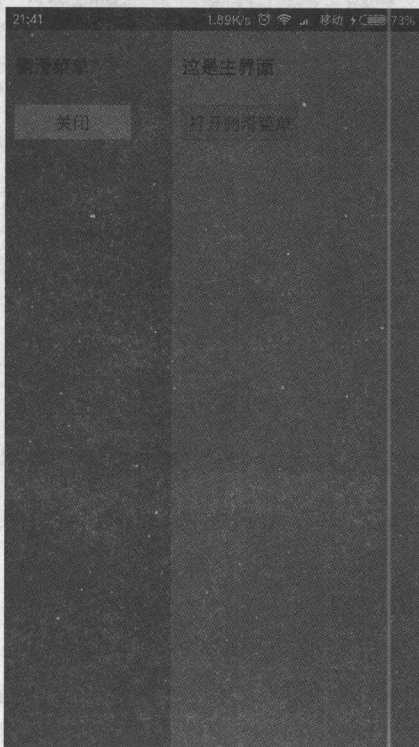


图 12-7 侧滑菜单效果

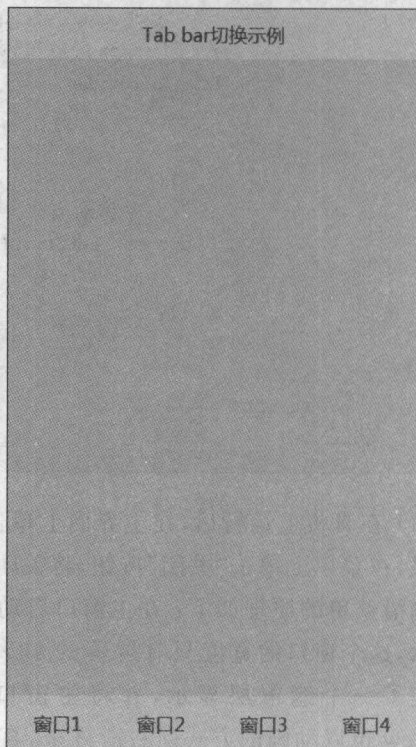


图 12-8 Tab bar 原始页面

(3) 打开 index.html, 页面代码修改如下所示:

```

<body>
  <header>
    Tab bar 切换示例
  </header>
  <footer>
    <span data-href="first.html" class="currentWv">窗口 1 </span>
    <span data-href="second.html">窗口 2 </span>
    <span data-href="third.html">窗口 3 </span>
    <span data-href="fourth.html">窗口 4 </span>
  </footer>
  <script type="text/javascript">

```

```

document.addEventListener('plusready', function() {
    var subpages = ['first.html',
        'second.html', 'third.html', 'fourth.html'];
    var subpage_style = {
        top: '45px',
        bottom: '51px'
    };
    var self = plus.Webview.currentWebview();
    var currentid = null;
    for (var i = 0; i < 4; i++) {
        var sub = plus.Webview.create(subpages[i],
            subpages[i], subpage_style);
        if (i > 0) {
            sub.hide();
        } else {
            currentid = subpages[i];
        }
        self.Append(sub);
    }
    document.querySelector("footer")
        .addEventListener("click", function(e) {
        var targetid = e.target.getAttribute("data-href");
        document.querySelector("span.currentWv")
            .classList.remove("currentWv");
        e.target.classList.add("currentWv");
        if (currentid != targetid) {
            plus.Webview.getWebviewById(currentid).hide();
            plus.Webview.getWebviewById(targetid).show();
            currentid = targetid;
        }
    })
});
</script>
</body>

```

(4) 真机运行后,出现了常见的底部 Tab 切换 Webview 的效果。

(5) Tab bar 切换 Webview 的原理如下:在主窗口启动时用 create 方法同时创建了 First、Second、Third、Fourth 4 个 Webview,它们的高度及在屏幕的上下位置恰好是除掉了主 Webview 的 header 和 footer 的高度和位置,如图 12-9 所示,并且这四个 Webview 都成了主 Webview 的子 Webview(这样,主 Webview 关闭的同时也会关闭所有的子 Webview),界面上只显示 First 的 Webview,其他 Webview 都隐藏了起来。当单击每个 Tab bar 时,再隐藏当前的 Webview,并自动切换到相应的 Webview。

【例 12-6】“闲鱼”App 的 Tab 突出菜单示例。

(1) 打开课件所附带的练习项目,在 Chrome 浏览器中以“移动设备模式”查看“index.html”“tabmenu.html”“menucontent.html”的页面效果,如图 12-10 所示。

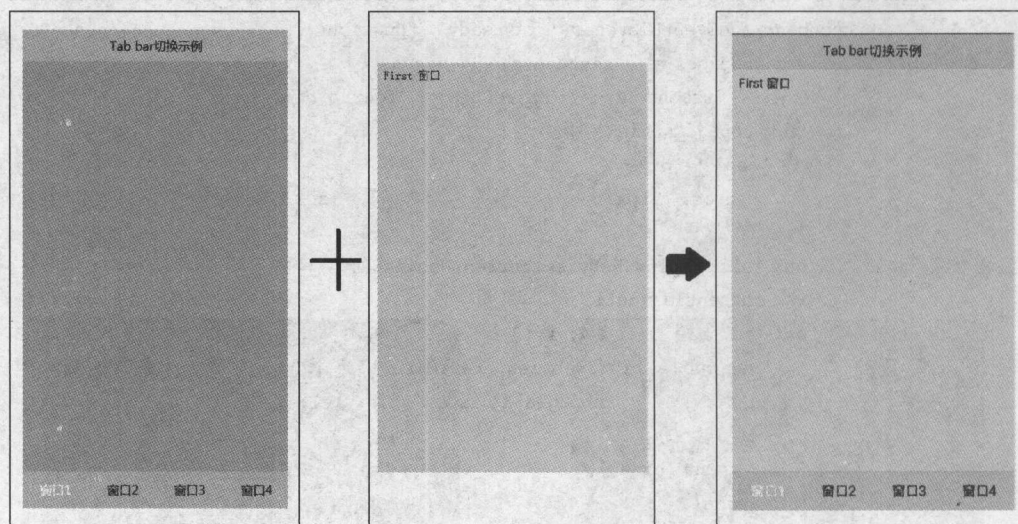


图 12-9 Tab bar 切换 Webview 原理

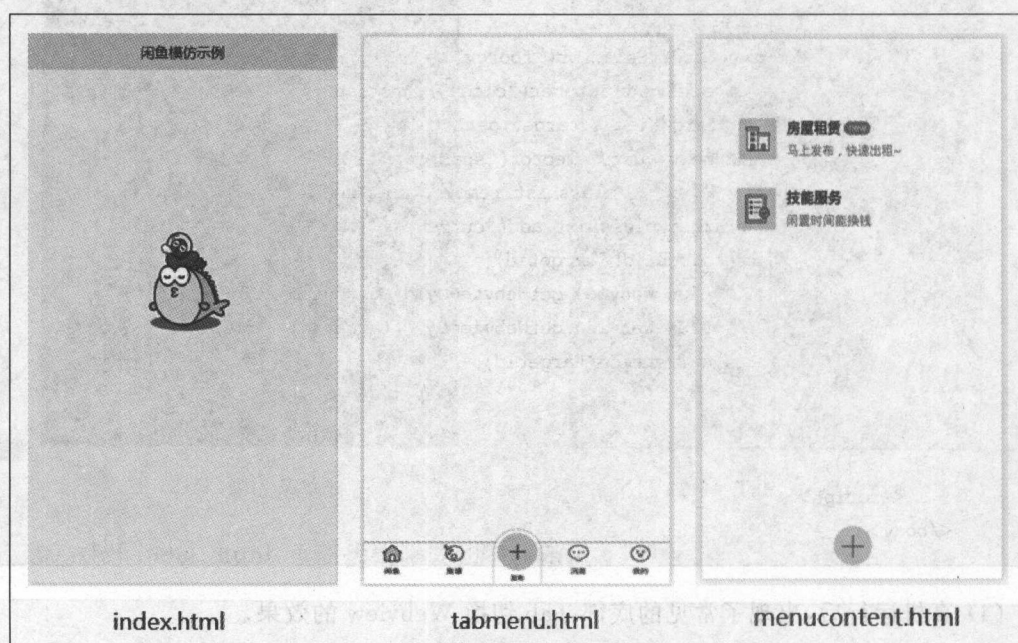


图 12-10 “闲鱼”效果模仿原始页面效果

(2) 打开 index.html, 修改其代码如下所示:

```
<body>
  <header>闲鱼模仿示例</header>
  <div id="content">
    
  </div>
```



```

<script>
    document.addEventListener('plusready', function() {
        var self = plus.Webview.currentWebview();
        //创建底部 Webview
        var tabview =
            plus.Webview.create("tabmenu.html", "tabmenu.html",
            {
                height: "70px",
                bottom: "0",
                background: "transparent",
                zIndex: 999
            });
        //添加作为子 Webview
        self.Append(tabview);
    });
</script>
</body>

```

(3) 打开 tabmenu.html, 修改其代码如下所示:

```

<body>
    <footer>
        
    </footer>
    <script>
        var mask = null; //遮罩 Webview 对象
        var menu = null; //menu 的 Webview 对象
        document.addEventListener('plusready', function() {
            document.getElementById('publish')
                .addEventListener('click', function() {
                    if(!mask) {
                        //创建一个 mask 遮罩并显示出来
                        mask = plus.Webview.create("", "mask", {
                            mask: "rgba(39,39,39,0.9)",
                            background: "transparent"
                        });
                        mask.addEventListener("maskClick", function(e) {
                            closeMenu();
                        });
                    }
                    mask.show();
                    //创建 menu 的 Webview
                    menu = plus.Webview.create("menucontent.html",
                        "menucontent.html", {
                        top: "0",
                        bottom: "0",
                        background: "transparent"
                    });
                });

```

```

        menu.show();
    });
    function closeMenu() {
        mask.hide();
    }
    });
</script>
</body>

```

(4) 打开 menucontent.html, 修改其代码如下所示:

```

<body scroll="no">
    <div id="decor1" class="decor">
    </div>
    <div id="decor2" class="decor">
    </div>
    <div id="footer">
        <div id="containter">
            <div id="yellowcontainer">
                
            </div>
        </div>
    </div>
    <script>
        var btnClose = null;
        document.addEventListener('plusready', function() {
            btnClose = document.getElementById("btnClose");
            if(btnClose) {
                btnClose.classList.add("rt");
                btnClose.addEventListener("click", function() {
                    closeContent();
                });
            }
            document.addEventListener("click", function() {
                closeContent();
            });
            function closeContent() {
                plus.Webview.getWebviewById("mask").hide();
                plus.Webview.currentWebview().close();
            }
        });
    </script>
</body>

```

这个模仿示例的原理简单解释如下: 如图 12-11 所示, 底部突出的菜单实际是放在一个 Webview 中的, 当主窗口启动时, 把 Webview2 的高度设置为 70px 和透明的, 与 Webivew1 叠加后(使用 zindex 属性控制层叠性)就可以看到类似“闲鱼”的突出菜单效果。当单击“发布”按钮时, 创建了一个空的透明 Webivew, 它主要是遮罩层, 再和

“menucontent.html”这个 Webview 叠加在一起,就形成了最终的效果,如图 12-12 所示。

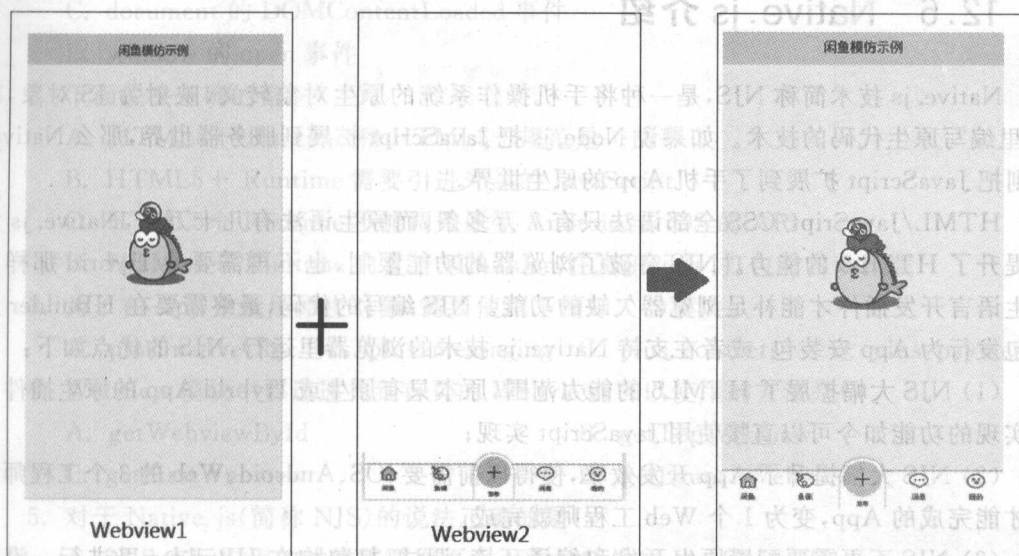


图 12-11 底部突出菜单的实现

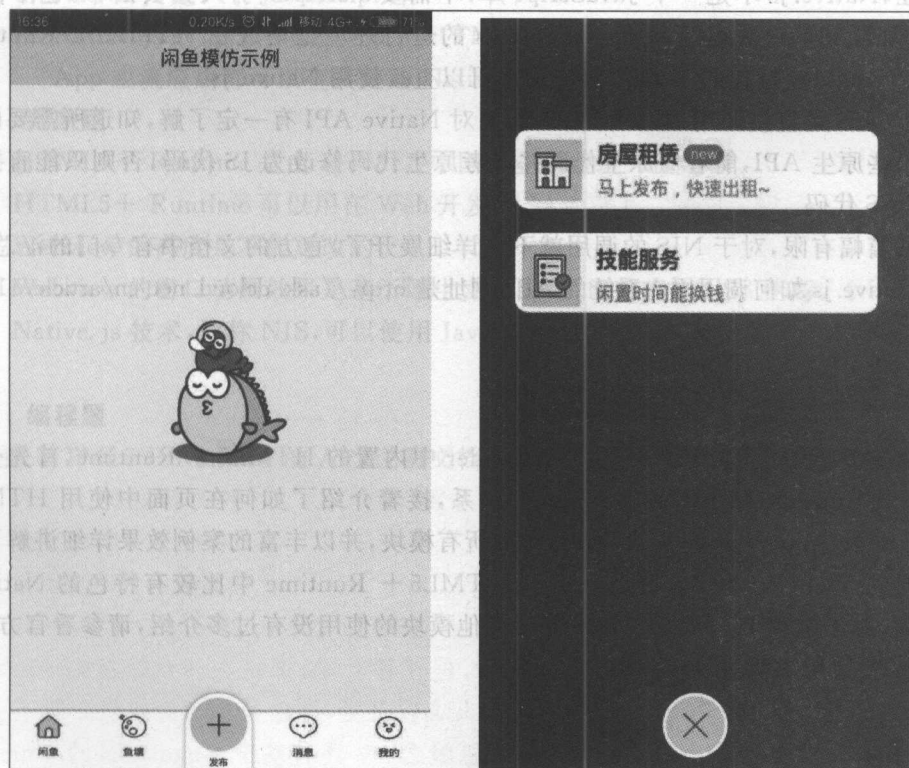


图 12-12 最终实现的效果

12.6 Native.js 介绍

Native.js 技术简称 NJS, 是一种将手机操作系统的原生对象转义, 映射为 JS 对象, 在 JS 里编写原生代码的技术。如果说 Node.js 把 JavaScript 扩展到服务器世界, 那么 Native.js 则把 JavaScript 扩展到了手机 App 的原生世界。

HTML/JavaScript/CSS 全部语法只有 7 万多条, 而原生语法有几十万条, Native.js 大幅提升了 HTML5 的能力。NJS 突破了浏览器的功能限制, 也不再需要像 Hybrid 那样由原生语言开发插件才能补足浏览器欠缺的功能。NJS 编写的代码, 最终需要在 HBuilder 里打包发行为 App 安装包, 或者在支持 Native.js 技术的浏览器里运行, NJS 的优点如下:

(1) NJS 大幅扩展了 HTML5 的能力范围, 原本只有原生或 Hybrid App 的原生插件才能实现的功能如今可以直接使用 JavaScript 实现;

(2) NJS 大幅提升了 App 开发效率, 使得以前需要 iOS、Android、Web 的 3 个工程师组队才能完成的 App, 变为 1 个 Web 工程师就完成。

(3) NJS 不再需要配置原生开发和编译环境, 调试、打包均在 HBuilder 里进行。没有 Mac 和 Xcode 一样可以开发 iOS 应用。

记住, Native.js 不是一个 JavaScript 库, 不需要 `<script>` 引入到页面中, 也不需要像 Node.js 那样有单独的运行环境。Native.js 的运行环境是集成在 HTML5+ Runtime 里的, 使用 HBuilder 打包的 App 或流应用都可以直接使用 Native.js。

由于 NJS 是直接调用 Native API, 需要对 Native API 有一定了解, 知道所需要的功能调用了哪些原生 API, 能看懂原生代码并参考原生代码修改为 JS 代码, 否则只能直接复制已有的 NJS 代码。

本书篇幅有限, 对于 NJS 的调用就不再详细展开了, 官方的文档中有专门的汇总文档, 示范了 Native.js 如何调用原生系统的 API, 网址是 <http://ask.dcloud.net.cn/article/114>。

小结

本章主要讲解了 DCloud 公司的 HBuilder 中内置的 HTML5+ Runtime, 首先介绍了 HTML5+ Runtime 和 HTML5+ 规范的关系, 接着介绍了如何在页面中使用 HTML5+ Runtime 开发 App, 并列举了 HTML5+ 的所有模块, 并以丰富的案例效果详细讲解了其中比较重要的 Webview 模块, 最后介绍了 HTML5+ Runtime 中比较有特色的 Native.js。限于篇幅, 对于本章 HTML5+ Runtime 其他模块的使用没有过多介绍, 请参看官方 Demo 示例和 API 使用说明进一步掌握。

习题

一、选择题

1. HTML5+ API 的使用必须在页面的()事件中完成。
A. body 的 onload 事件

- ## 二、判断题

- ()

对照 HTML5+ Runtime 的 API 说明文档和官方 Demo 熟悉各模块的使用。

第 13 章

CHAPTER 13

MUI 框架

学习目标

- 了解 MUI 框架的使用。
- 掌握 MUI 项目页面的布局和一些特殊使用。
- 掌握 MUI 框架一些内置方法和对象的使用。
- 熟练掌握 MUI 框架中的事件管理。
- 熟练掌握 MUI 框架中的窗口管理。
- 掌握 MUI 框架中各种 UI 组件的使用。
- 了解 MUI 各插件的使用。
- 熟练掌握 MUI 框架中 AJAX 通信的各方法。
- 熟练掌握在 Chrome 中调试 Android 应用。

作为高性能的 App 框架, MUI 提供了一系列 UI 组件、内置的对象和方法, 为 HTML5 App 的快速开发提供了可能。本章针对 MUI 框架使用的各方面作了详细讲解, 熟悉 MUI 对于 HTML5 App 的开发有很大的帮助。借助 MUI 框架, 再结合第 12 章的 HTML5+ Runtime, 快速完成一个移动 App 并非难事。

13.1 MUI 介绍

性能和体验的差距, 一直是移动应用开发者放弃或怀疑 HTML5 的首要原因。浏览器天生的切页白屏、不忍直视的转页动画、浮动元素的抖动、无法流畅下拉刷新、侧滑抽屉卡顿等问题, 都让 HTML5 开发者倍感挫败, 尤其程序在 Android 低端机运行时的性能根本无法忍受。另一方面, 浏览器默认控件的样式又少又丑, 制作一个漂亮的控件非常麻烦。

在 MUI 之前, 业界也出现过不少移动 App 框架, 例如基于 jQuery 的 jQuery Mobile 框架, 虽然 jQuery 在 PC 端浏览器中表现优异, 但在移动端, 即使在相对较新的 Android 和 iOS 硬件上, jQuery Mobile 的性能也低得让人难以忍受, 这也是本书为何只介绍了 jQuery 的核心语法, 而没有介绍 jQuery Mobile 的原因之一; 还有基于 AngularJS 的 ionic 框架, AngularJS 设计本身是为了 PC 端网页的双向数据绑定, 结果引入到移动 App 中了; Bootstrap 这种响应式设计, 性能在低端机不足, 而且 UI 风格与 App 的相差甚远; Framework7 主要针对 iPhone, 并不能兼容所有设备。于是为了方便广大开发者, DCloud 公司基于 Ratchet 框架制作了 MUI。

MUI是一款高性能 App 框架,也是目前最接近原生 App 效果的 UI 框架。MUI 中的 UI 组件设计是以 iOS 7 为基础,补充了部分 Android 特有控件。MUI 框架有效地解决了原有框架的一些问题,可以很方便地开发出高性能 App。MUI 的定位是:最接近原生体验的移动 App 的 UI 框架。正是这样的定位,产生了 MUI 的几个特点:轻、小、只涉及 UI、只为移动 App 而生、界面风格原生化。MUI 不同于 jQuery,没有封装 DOM 操作,不做与 UI 无关的事情,开发人员也可以使用 jQuery Mobile 或 Zepto 这样的框架,在开发中并不冲突,但 DCloud 公司建议不使用其他框架,原因有两点:

- 性能问题,层层封装的框架,尤其是遍历循环 DOM 时,影响效率,特别是在低端 Android 手机上。
- 兼容性问题,jQuery 的成功主要是 PC 端的浏览器兼容性,PC 上的浏览器性能好,即使用 jQuery Mobile 也不会影响,但它根本就不是为手机设计的。

手机上(Android 和 iOS)的浏览器都是 Webkit 核心,根本不需要考虑兼容性了,而且 HBuilder 提供了代码块来简化开发,非常快捷方便,而且目前原生的 JavaScript 执行在理论上可以获得最高性能。

13.2 MUI 的示例

在网址 <http://www.DCloud.io/mui.html> 上,DCloud 公司提供了一个 MUI 的在线 Demo(如果你的计算机屏幕是支持触摸屏的,可以单击浏览各 MUI 中提供的各 UI 组件效果),如图 13-1 所示。页面上也有个二维码,使用手机扫描后可以直接下载示例的手机 App 程序,在手机上安装后可体验 MUI 的各种 UI 效果;直接单击二维码也可以打开它的 Demo 页面,请在 Chrome 中切换到“移动设备模式”进行体验。



图 13-1 MUI 的在线 Demo

为了方便开发者迅速掌握各种 UI 的使用, HBuilder 内含了 MUI 的 Demo 演示的所有源代码,使用的方法是打开 HBuilder,单击“文件”菜单中的“新建”命令,选择“移动 App”后,弹出对话框,如图 13-2 所示,选择模板“Hello mui”,输入应用名称,选择存放位置后,单击“完成”按钮,即可创建 MUI 的 Demo 演示项目。

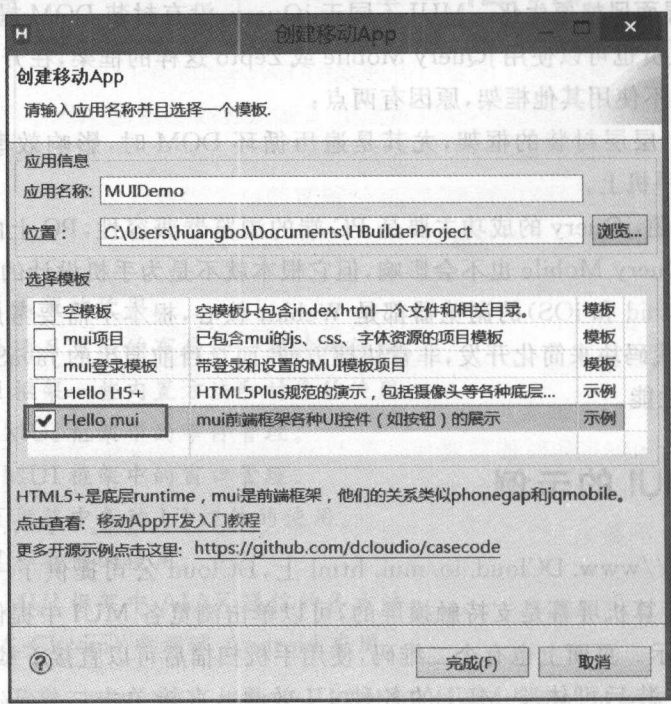


图 13-2 创建 MUI Demo 项目

每次 MUI 升级(修改或增加 UI 控件)后, HBuilder 会自动更新, 这个包含的 Demo 项目也会自动升级。

13.3 使用 MUI

要使用 MUI 开发 HTML5 App,首先必须创建“MUI”项目,使用的方法是打开 HBuilder,单击“文件”菜单中的“新建”命令,选择“移动 App”后,将弹出对话框,如图 13-3 所示,选择模板“mui 项目”,输入应用名称,选择存放位置后,单击“完成”按钮,即可创建一个 MUI 项目,这个项目会自动包含 MUI 所用到的. js,. css 和字体资源等文件。

MUI 项目中的各 HTML 页面,如果要使用 MUI 框架,页面必须引入 MUI 的核心类库和对应的. css 文件。为了简化操作,HBuidler 使用了内置模板的方式。使用的方法是,选中开发的项目,点击鼠标右键,选择“新建”命令,再选择“HTML 文件”,将弹出“创建文件向导”对话框,如图 13-4 所示,选择模板“含 mui 的 html”,输入文件名,再单击“完成”按钮,就可以创建一张包含 MUI 的 HTML 页面。

创建页面后,来简单看看这样的页面有何特点。下面是这种页面的示范代码,从这个示

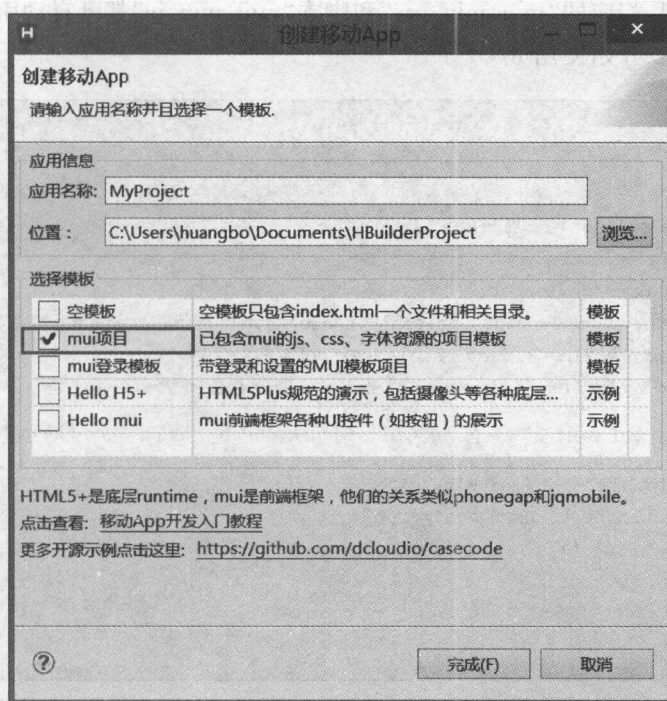


图 13-3 创建 MUI 项目

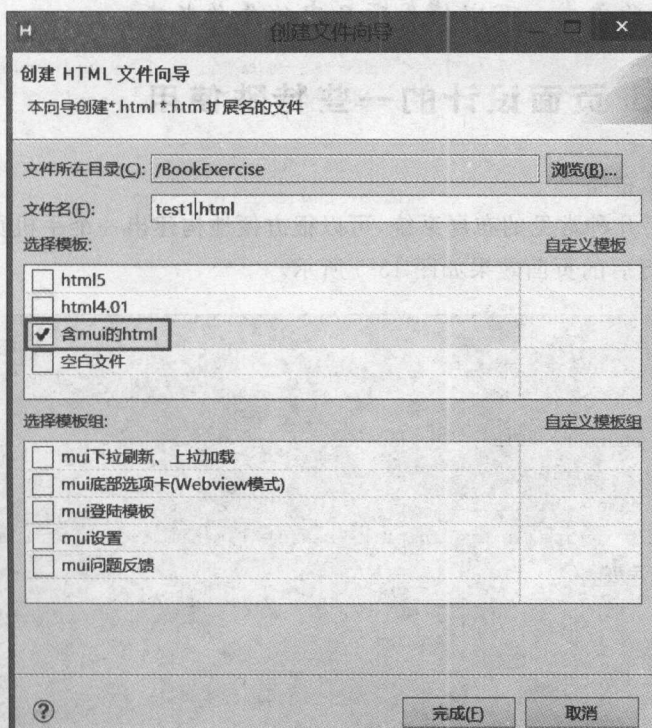


图 13-4 创建 MUI 页面

范代码中可以看到,相应的“mui.min.css”和脚本“mui.min.js”都已自动引入。在这样准备就绪的页面上,可以开始使用 MUI 了。

```
<!doctype html>
<html>
  <head>
    <meta charset = "UTF-8">
    <title></title>
    <meta name = "viewport"
content = "width = device - width, initial - scale = 1, minimum - scale = 1,
maximum - scale = 1, user - scalable = no" />
    <link href = "css/mui.min.css" rel = "stylesheet" />
  </head>
  <body>
    <script src = "js/mui.min.js"></script>
    <script type = "text/javascript">
      mui.init()
    </script>
  </body>
</html>
```



MUI 完全开源,在实际开发中,.css 和 .js 文件都使用压缩版本,如果有修改或定制的需求,可以使用项目中的原始版本。

13.4 MUI 页面设计的一些特殊使用

1. 页面的整体布局

MUI 中提供了几种常见的布局系统,可以很方便地构建出一个手机页面。先来看一个最简单的例子,运行后的页面效果如图 13-5 所示。

```
<!doctype html>
<html>
  <head>
    <meta charset = "UTF-8">
    <title></title>
    <meta name = "viewport"
content = "width = device - width, initial - scale = 1, minimum - scale = 1, maximum - scale = 1,
user - scalable = no" />
    <link href = "css/mui.min.css" rel = "stylesheet" />
  </head>
  <body>
    <header class = "mui-bar mui-bar-nav">
      <h1 class = "mui-title">标题</h1>
    </header>
    <footer class = "mui-bar mui-bar-footer">
```

```

        底部
    </footer>
    <div class = "mui-content">
        主体
    </div>
    <script src = "js/mui.min.js"></script>
    <script type = "text/javascript">
        mui.init()
    </script>
</body>
</html>

```

当底部内容为选项卡的时候,会将. mui-bar-footer 替换为. mui-bar-tab,示例代码如下:

```

<!doctype html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title></title>
        <meta name = "viewport"
content = "width = device-width, initial-scale = 1, minimum-scale = 1, maximum-scale = 1,
user-scalable = no" />
        <link href = "css/mui.min.css" rel = "stylesheet" />
    </head>
    <body>
        <header class = "mui-bar mui-bar-nav">
            <h1 class = "mui-title">标题</h1>
        </header>
        <nav class = "mui-bar mui-bar-tab">
            <a class = "mui-tab-item mui-active">
                <span class = "mui-icon mui-icon-home"></span>
                <span class = "mui-tab-label">首页</span>
            </a>
            <a class = "mui-tab-item">
                <span class = "mui-icon mui-icon-phone"></span>
                <span class = "mui-tab-label">电话</span>
            </a>
            <a class = "mui-tab-item">
                <span class = "mui-icon mui-icon-email"></span>
                <span class = "mui-tab-label">邮件</span>
            </a>
            <a class = "mui-tab-item">
                <span class = "mui-icon mui-icon-gear"></span>
                <span class = "mui-tab-label">设置</span>
            </a>
        </nav>
        <div class = "mui-content">
            主体

```

```

        </div>
        <script src = "js/mui.min.js"></script>
        <script type = "text/javascript">
            mui.init()
        </script>
    </body>
</html>

```

运行后的页面效果如图 13-6 所示。

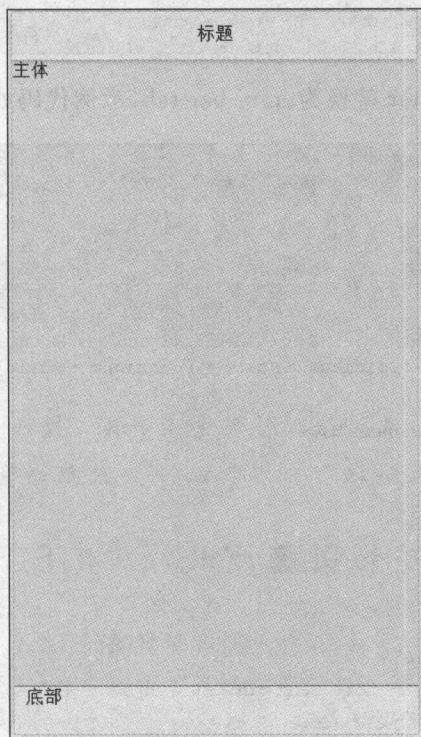


图 13-5 简单的布局效果

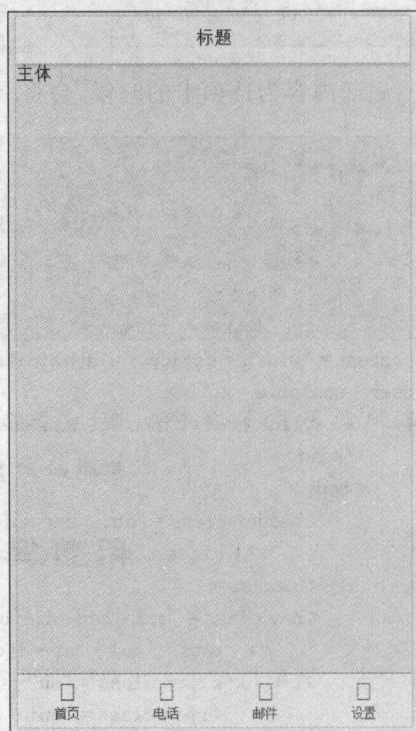


图 13-6 tab 页效果

在使用时要注意这两个细节：

(1) 固定栏靠前, 所谓 MUI 的固定栏, 也就是 class 属性带有“mui-bar”属性的 HTML 元素节点, 它们都是基于 fixed 定位的元素。常见组件包括: 顶部导航栏 (HTML 元素的 class 属性中包含了“mui-bar-nav”), 底部工具条 (class 属性中包含了“mui-bar-footer”), 底部选项卡 (class 属性中包含了“mui-bar-tab”)。它们一定要放在 class 属性包含“mui-content”的 HTML 元素之前, 即使是底部工具条和底部选项卡, 也要放在之前, 否则固定栏会遮住部分主内容。因为固定栏是基于 fixed 定位, 不受流式布局限制, 普通内容依然会从 top:0 的位置开始布局, 这样就会被固定栏遮罩。

(2) 为了使用简便, 建议将除固定栏之外的所有内容, 全部放在 class 属性为“mui-content”的容器中。



对于顶部导航栏可以在页面中输入“mHeader”进行快捷生成, 底部

选项卡可以输入“mTab”进行快捷生成,输入时不需要切换到大写字母,更多快捷键请参考链接 <http://dev.DCloud.net.cn/mui/snippet/#snippet>。

2. 内联块元素和块级元素的设置

如果需要把 HTML 元素快速转换成内联块元素,可以给元素 class 属性添加“mui-inline”类,也就是设置 CSS 属性 display:inline-block,将元素呈递为内联对象,但是对象的内容作为块对象呈递,示例如下:

```
<div>
  <div class="mui-inline">苹果</div>
  <div class="mui-inline">香蕉</div>
  <div class="mui-inline">桔子</div>
</div>
```

同样地,若需要把 HTML 元素快速转换成块级元素,可以给元素 class 属性添加“mui-block”类,也就是设置 CSS 属性 display:block,示例如下:

```
<div>
  <span class="mui-block">苹果</span>
  <span class="mui-block">香蕉</span>
  <span class="mui-block">桔子</span>
</div>
```

3. 浮动

为了简化页面设计中常用的浮动设计,MUI 的配套.css 中已经设置好了左浮动和右浮动,以及清除浮动的样式,可以给元素 class 属性值设置“mui-pull-left”实现左浮动或者设置“mui-pull-right”实现右浮动,设置“mui-clearfix”清除浮动。示例如下:

```
<div class="mui-pull-left">
  左浮动
</div>
<div class="mui-pull-right">
  右浮动
</div>
<div class="mui-clearfix">
  清除浮动
</div>
```

4. 列表元素的处理

ul 标签在浏览器中显示时,会有默认的原点和内边距,可以给 ul 元素的 class 属性设置“mui-list-unstyled”去掉,设置“mui-list-inline”将 li 标签元素设置为内联块元素。

```
<ul class="mui-list-unstyled mui-list-inline">
  <li>Item1</li>
```

```
<li>Item2 </li>
<li>Item3 </li>
</ul>
```

5. 元素的显示与隐藏

可以给 HTML 元素的 class 属性设置“mui-visibility”让元素可见, class 属性设置“mui-hidden”隐藏元素。

6. OS 环境多平台适配

MUI 会通过 .mui.os.* 方法判断环境。将 CSS 的样式选择器 .mui-plus,.mui-ios,.mui-plus-stream,.mui-android,.mui-wechat,.mui-ios-version,.mui-android-version,.mui-wechat-version 等绑定在 HTML 元素的 class 属性中,可以通过这些样式类作出当前的运行判断,还可以作出一些适配,例如:

```
<div class="mui-plus-visible">
  <input type="text" value="在 web 环境下隐藏,5+ 环境下显示">
</div>
<div class="mui-plus-hidden">
  <input type="text" value="在 web 环境下显示,5+ 环境下隐藏">
</div>
```

MUI 中默认在 plus 环境下和微信环境下设置了几个 class 选择器样式:

- “mui-plus-visible”: 在 HTML5+ 环境下显示,其他环境下隐藏;
- “mui-wechat-visible”: 在微信环境下显示,其他环境下隐藏;
- “mui-plus-hidden”: 在 HTML5+ 环境下隐藏,其他环境下显示;
- “mui-wechat-hidden”: 在微信环境下隐藏,其他环境下显示。



和 HTML5+ 规范不一样, MUI 既可以用于移动 App 开发上,也可以在网页中使用。

13.5 mui 对象的内置方法和对象

MUI 框架在设计中明显借鉴了 jQuery 的一些做法,但又不完全相同,如不提供任何 DOM 操作方法,下面简单介绍它的内置方法。

1. mui() 方法

mui() 方法使用 css 选择器获取 HTML 元素对象,返回一个 mui 对象数组。例如:

```
mui("p"); //选取所有<p>元素
mui("p.title") //选取所有包含.title类的<p>元素
```

若要将 mui 对象转化成 DOM 对象,可使用如下方法(类似于 jQuery 对象转成 DOM 对象):

```
//obj1 是 mui 对象
var obj1 = mui("#title");
//obj2 是 DOM 对象
var obj2 = obj1[0];
```

2. each()方法

each 既是一个类方法,同时也是一个对象方法,两种方法的适用场景不同;换言之,可以使用 mui.each()去遍历数组或 json 对象,也可以使用 mui(selector).each()去遍历 DOM 结构,这一点与 jQuery 的 each 方法相同。

3. init()方法

MUI 框架将很多功能配置都集中在 mui.init()方法中,要使用某项功能,只需要在 mui.init()方法中完成对应参数配置即可,目前支持在 mui.init 方法中配置的功能包括创建子页面、关闭页面、手势事件配置、预加载、下拉刷新、上拉加载、设置系统状态栏背景颜色,配置形式如下(这里只列出各配置模块所有可配置项):

```
mui.init({
  //子页面
  subpages: [{
    //...
  }],
  //预加载
  preloadPages:[
    //...
  ],
  //下拉刷新、上拉加载
  pullRefresh: {
    //...
  },
  //手势配置
  gestureConfig:{
    //...
  },
  //侧滑关闭
  swipeBack:true,           //Boolean(默认 false)启用右滑关闭功能
  //监听 Android 手机的 back、menu 按键
  keyEventBind: {
    backbutton: false,       //Boolean(默认 true)关闭 back 按键监听
    menubutton: false        //Boolean(默认 true)关闭 menu 按键监听
  },
  //处理窗口关闭前的业务
  beforeback: function() {
    //...
  },
  //设置状态栏颜色
  statusBarBackground: '#9defbcg', //设置状态栏颜色,仅 iOS 可用
  preloadLimit:5//预加载窗口数量限制(一旦超出,先进先出)默认不限制
})
```




MUI 需要在页面加载时初始化很多基础控件,如监听返回键,因此务必在每个页面中调用该方法。

4. scrollTo()方法

滚动窗口屏幕到指定位置,该方法是对 window.scrollTo()方法在手机端的增强实现,可设定滚动动画时间及滚动结束后的回调函数,鉴于手机屏幕大小,该方法仅可实现屏幕纵向滚动,它的语法结构为:

```
mui.scrollTo(ypos[,duration][,handler])
```

ypos 是整型值,要在窗口文档显示区左上角显示的文档的 y 坐标。

duration 是滚动时间周期,单位是毫秒。

handler 是滚动结束后执行的回调函数。

例如,1 秒钟之内滚动到页面顶部:

```
mui.scrollTo(0,1000);
```

5. mui.os 对象

在浏览器中,经常需要通过 navigator.userAgent 判断当前运行环境,MUI 框架对此进行了类似的封装,它的属性如表 13-1 所示。

表 13-1 mui.os 对象属性说明

属 性	说 明
mui.os.plus	Boolean,返回是否在基座中运行
mui.os.stream	Boolean 类型,返回是否为流应用
mui.os.android	Boolean 类型,返回是否为 Android
mui.os.ios	Boolean 类型,返回是否为 iOS
mui.os.ipad	Boolean 类型,返回是否为 iPad
mui.os.iphone	Boolean 类型,返回是否为 iPhone
mui.os.wechat	Boolean 类型,返回是否在微信中运行
mui.os.version	String 类型,当前运行环境的版本号

13.6 事件管理

1. tap 事件

快速响应是移动应用开发实现的重中之重,研究表明,当延迟超过 100ms,用户就能感受到界面的卡顿,然而手机浏览器的 click 单击存在 300ms 延迟,MUI 为了解决这个问题,封装了 tap 事件,因此在任何点击的时候,请忘记 click 及 onclick 操作,统一使用如下标准的事件监听函数代码形式:

```
element.addEventListener('tap',function(){
    //单击响应逻辑
},false);
```

2. 事件绑定

除了可以使用 `addEventListener()` 方法监听某个特定元素上的事件外,也可以使用 `on()` 方法实现批量元素的事件绑定,它的语法格式为:

```
mui(selector1).on(event,selector2,handler);
```

`event` 是 `String` 类型,代表要监听的事件名称,如“tap”。

`handler` 是事件触发时的回调函数,可以通过事件参数对象获得事件详情。

`selector1` 和 `selector2` 都是选择器,但是 `selector2` 必须是 `selector1` 代表的 HTML 元素对象下面的后代选择器。

示例代码如下:

```
mui(".mui-table-view").on('tap','.mui-table-view-cell',function(){
    //监听选择器".mui-table-view"对象下面的子元素选择器
    ".mui-table-view-cell"对象的 tap 事件
})
```

3. 事件取消

使用 `on()` 方法绑定事件后,若希望取消绑定,则可以使用 `off()` 方法。根据传入参数的不同,`off()` 方法有不同的实现逻辑。

- `mui(selector).off()`: 删除 `mui` 对象上所有事件。
- `mui(selector1).off(event,selector2)`: 删除 `selector1` 选择器对应的 HTML 对象中的 `selector2` 对应的子对象上特定事件的所有回调。
- `mui(selector1).off(event,selector2,handler)`: 删除 `selector1` 选择器对应的 HTML 对象中的对应的 `selector2` 子对象上之前绑定的事件函数,不支持匿名函数。

4. 事件触发

使用 `mui.trigger()` 方法可以动态触发特定 DOM 元素上的事件。它的语法形式为:

```
mui.trigger(element,event,data);
```

`element` 是触发事件的 DOM 对象。

`event` 是事件名称,如“tap”。

`data` 是 JSON 对象格式,代表需要传递给事件的业务参数。

例如,下面的代码能自动触发按钮的点击事件:

```
var btn = document.getElementById("btnSend");
//监听点击事件
```

```
btn.addEventListener("tap",function () {
    console.log("tap event trigger");
});
//自动触发按钮的点击事件
mui.trigger(btn, 'tap');
```



部分 mui 控件监听的事件无法通过 mui.trigger 触发，例如无法实现自动触发 mui 返回图标，实现关闭当前页面的功能，该部分逻辑正在优化中。

5. 手势事件

在开发移动端的应用时，会用到很多的手势操作，例如滑动、长按等，为了方便开发者快速集成这些手势，mui 内置了常用的手势事件，目前支持的手势事件见如表 13-2 所示。

表 13-2 MUI 手势事件说明

分类	事件名称	说明
点击	“tap”	单击屏幕
	“doubletap”	双击屏幕
长按	“longtap”	长按屏幕
	“hold”	按住屏幕
	“release”	离开屏幕
滑动	“swipeleft”	向左滑动
	“swiperight”	向右滑动
	“swipeup”	向上滑动
	“swipedown”	向下滑动
拖动	“dragstart”	拖动开始
	“drag”	拖动中
	“dragend”	拖动结束

根据使用频率，MUI 默认会监听部分手势事件，如点击、滑动事件；为了开发出更高性能的 App，MUI 支持用户根据实际业务需求，通过 mui.init() 方法中的 gestureConfig 参数，配置具体需要监听的手势事件，配置如下：

```
mui.init({
    gestureConfig:{
        tap: true,           //默认为 true
        doubletap: true,    //默认为 false
        longtap: true,      //默认为 false
        swipe: true,        //默认为 true
        drag: true,         //默认为 true
        hold: false,        //默认为 false,不监听
        release: false      //默认为 false,不监听
    }
});
```


6. 自定义事件

窗口可以添加自定义事件,监听操作和标准 JS 事件监听类似,可直接通过 window 对象添加,如下:

```
window.addEventListener(customEventName, function(event){
    //通过 event.detail 可获得传递过来的参数内容
});
```

为了触发窗口中的自定义事件,MUI 封装出了一个 mui.fire()方法,它的语法形式如下:

```
mui.fire(target, event, data);
```

target 是目标 Webview 对象。

event 是自定义事件名称。

data 是个 JSON 对象,用于传值。

13.7 窗口管理

1. HTML5+初始化

在 App 开发中,若要使用 HTML5+扩展 API,必须等 plusready 事件发生后才能正常使用,MUI 框架将该事件封装成了 mui.plusReady()方法,在涉及 HTML5+ API 调用时,建议都写在 mui.plusReady 方法中。如下为打印当前页面 URL 的示例:

```
mui.plusReady(function(){
    console.log("当前页面 URL: " + plus.webview.currentWebview().getURL());
});
```

2. 打开新窗口

制作 HTML5 App 时,一个无法避开的问题就是转场动画;Web 是基于超链接构建的,从一个页面点击链接跳转到另一个页面,如果通过有刷新的打开方式,用户要面对一个空白的页面等待;如果通过无刷新的方式,用 JavaScript 移入 DOM 节点(常见的单页应用解决方案),会碰到很高的性能挑战:DOM 节点繁多,页面太大,转场动画不流畅甚至导致浏览器崩溃。MUI 的解决思路是:单 Webview 只承载单个页面的 DOM,减少 DOM 层级及页面大小,页面切换使用原生动画,将最耗性能的部分交给原生实现。打开新窗口的语法形式为:

```
mui.openWindow({
    url:new - page - url,
    id:new - page - id,
```

```

styles:{//和HTML5+中Webview的open方法中的配置一致},
extras:{
    ...//自定义扩展参数,可以用来处理页面间传值
},
createNew:false,//是否重复创建同样id的webview,默认为false
show:{
    autoShow:true,//页面loaded事件发生后自动显示,默认为true
    aniShow:animationType,//页面显示动画,默认为"slide-in-right";
    duration:animationTime//页面动画持续时间,Android平台默认100毫秒,iOS平台默认200
    毫秒;
},
waiting:{
    autoShow:true,//自动显示等待框,默认为true
    title:'正在加载...',//等待对话框上显示的提示内容
    options:{
        width:waiting-dialog-width,//等待框背景区域宽度,默认根据内容自动计算合适宽度
        height:waiting-dialog-height,//等待框背景区域高度,默认根据内容自动计算合适
        高度
        ...
    }
}
})

```

这里要注意的是,MUI框架在打开新窗口时等待框的处理逻辑为:显示等待框→创建目标页面 Webview→目标页面 loaded 事件发生→关闭等待框。因此,只有当新页面为新创建页面(Webview)时,才会显示等待框,否则,若为预加载好的页面,则直接显示目标页面,不会显示等待框。

3. 关闭窗口

MUI框架将窗口关闭功能封装在 mui.back 方法中,具体执行逻辑是:若当前 Webview 为预加载页面,则 hide 当前 Webview; 否则,close 当前 Webview。

在 MUI 框架中,有三种操作会触发页面关闭(执行 mui.back 方法):

(1) 点击 class 属性包含“mui-action-back”类的控件。

若希望在顶部导航栏之外的其他区域添加关闭页面的控件,只需要在对应 HTML 元素的 class 属性上添加“mui-action-back”类即可,如下为一个关闭按钮示例:

```
<button type="button" class='mui-btn mui-btn-danger mui-action-back'>关闭</button>
```

(2) 在屏幕内,向右快速滑动。

MUI框架封装的窗口右滑关闭功能,默认未启用,若要使用右滑关闭功能,需要在 mui.init();方法中设置 swipeBack 参数,如下:

```

mui.init({
    swipeBack:true //启用右滑关闭功能
});

```

(3) Android 手机按下 back 按键: MUI 框架默认会监听 Android 手机的 back 按键, 然后执行页面关闭逻辑; 若不希望 MUI 自动处理 back 按键, 可通过如下方式关闭 MUI 的 back 按键监听:

```
mui.init({
  keyEventBind: {
    backbutton: false //关闭 back 按键监听
  }
});
```

除了如上三种操作外, 也可以直接调用 `mui.back()` 方法, 执行窗口关闭逻辑; `mui.back()` 仅处理窗口逻辑, 若希望在窗口关闭之前再处理一些其他业务逻辑, 则可将业务逻辑抽象成一个具体函数, 然后注册为 `mui.init` 方法的 `beforeback` 参数。 `beforeback` 的执行逻辑为: 若返回 `false`, 则不再执行 `mui.back()` 方法; 否则返回 `true` 或无返回值, 继续执行 `mui.back()` 方法。需要注意的是: `beforeback` 的执行返回必须是同步的(阻塞模式), 若使用 `nativeUI` 这种异步 `js`(非阻塞模式), 则可能会出现意想不到的结果。例如, 通过 `plus.nativeUI.confirm()` 弹出确认框, 可能用户尚未选择, 页面已经返回了(`beforeback` 同步执行完毕, 无返回值, 继续执行 `mui.back()` 方法, `nativeUI` 不会阻塞 `js` 进程)。在这种情况下, 若要自定义业务逻辑, 就需要复写 `mui.back` 方法了。如下为一个自定义示例, 每次都需要用户确认后, 才会关闭当前页面。

```
//备份 mui.back, mui.back 已将窗口关闭逻辑封装得比较完善(预加载及父子窗口)
var old_back = mui.back;
mui.back = function(){
  var btn = ["确定", "取消"];
  mui.confirm('确认关闭当前窗口?', 'Hello MUI', btn, function(e){
    if(e.index == 0){
      //执行 mui 封装好的窗口关闭逻辑;
      old_back();
    }
  });
};
```

4. 预加载

所谓的预加载技术就是在用户尚未触发窗口跳转时, 提前创建目标窗口, 这样当用户跳转时, 就可以立即进行页面切换, 节省创建新页面的时间, 提升 App 使用体验。 MUI 框架提供了两种方式实现页面预加载。

(1) 通过 `mui.init()` 方法中的 `preloadPages` 参数进行配置, 例如:

```
mui.init({
  preloadPages:[
    {
      url:prelaod~page~url, //页面路径
      id:preload~page~id,  //Webview 的 id
    }
  ]
});
```



```

        styles:{},          //窗口参数
        extras:{},         //自定义扩展参数
        subpages:[{}],{}   //预加载页面的子页面
    }
    ],
    preloadLimit:5          //预加载窗口数量限制(一旦超出,先进先出)默认不限制
});

```

该种方案使用简单、可预加载多个页面,但不会返回预加载每个页面的引用,若要获得对应 Webview 引用,还需要通过 `plus.webview.getWebviewById` 方式获得;另外,因为 `mui.init()` 方法是异步执行,在没有预加载完成前,去获得对应 Webview 引用,可能会失败。

(2) 通过 `mui.preload()` 方法预加载,例如:

```

var page = mui.preload({
    url:new - page - url, //页面路径
    id:new - page - id,   //Webview 的 id
    styles:{},           //窗口参数
    extras:{}             //自定义扩展参数
});

```

通过这种方法预加载,可立即返回对应 Webview 的引用,但一次仅能预加载一个窗口,若需加载多个 Webview,则需多次调用 `mui.preload()` 方法。

5. 子页面

在移动应用开发过程中,经常遇到“卡头卡尾”的页面,此时若使用局部滚动,在 Android 手机上会出现滚动不流畅的问题。MUI 框架的解决思路是:将需要滚动的区域通过单独的 Webview 实现,完全使用原生滚动。具体做法则是:将目标页面分解为主页面和内容页面,主页面显示卡头卡尾区域,例如顶部导航、底部选项卡等;内容页面显示具体需要滚动的内容,然后在主页面中调用 `mui.init` 方法初始化内容页面。例如:

```

mui.init({
    subpages:[{
        url:your - subpage - url,          //子页面地址,支持本地地址和网络地址
        id:your - subpage - id,            //子页面 id
        styles:{
            top:subpage - top - position,   //子页面顶部位置
            bottom:subpage - bottom - position, //子页面底部位置
            width:subpage - width,          //子页面宽度,默认为 100 %
            height:subpage - height,        //子页面高度,默认为 100 %
            ...
        },
        extras:{}                            //额外扩展参数
    }]
});

```

如图 13-7 所示效果, `index.html` 的作用就是显示固定导航, `list.html` 显示具体列表内

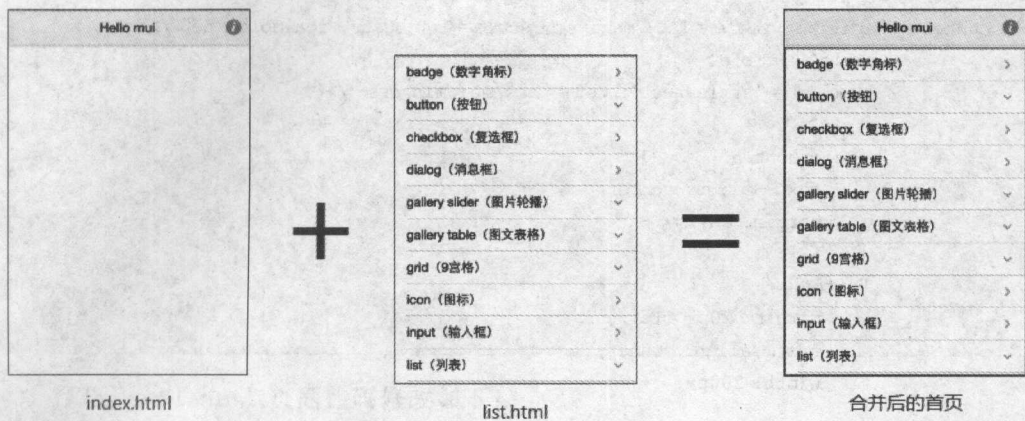


图 13-7 创建子页面效果

容,列表项的滚动是在 list.html 所在 Webview 中使用原生滚动,既保证了滚动条不会穿透顶部导航,符合 App 的体验,也保证了列表流畅滚动,解决了区域滚动卡顿的问题。

创建代码比较简单,如下:

```
mui.init({
  subpages:[{
    url:'list.html',
    id:'list.html',
    styles:{
      top:'45px', //mui 标题栏默认高度为 45px;
      bottom:'0px'//默认为 0px,可不定义;
    }
  }]
});
```

6. 窗口之间数据的传递

无论是在网页开发中,还是在 App 开发中,都有很多场景需要在窗口之间传递数据。MUI 框架中窗口之间传递数据分为两种情况:(1)使用 mui.openWindow 打开新窗口的同时传递数据;(2)向已预加载的窗口传递数据。下面用一个例子来演示如何针对这两种情况传递数据。

【例 13-1】窗口之间数据的传递示例。

(1) 新建“移动 App”项目,模板选择“mui 项目”,成功后在项目中新建两个 HTML 页面 new01.html 和 new02.html,模板选择“含 mui 的 html”。

(2) 打开 index.html,页面代码修改如下:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
```



```

content = "initial - scale = 1.0, maximum - scale = 1.0, user - scalable = no" />
<title></title>
<link rel = "stylesheet" href = "css/mui.min.css" />
<style>
    .mui-content{
        text-align: center;
        margin: 0 auto;
    }
    button{
        margin: 20px auto;
        text-align: center;
        width: 200px;
    }
</style>
</head>
<body>
    <header class = "mui-bar mui-bar-nav">
        <h1 class = "mui-title">窗口之间传递数据示例</h1>
    </header>
    <div class = "mui-content">
        <button id = "btnOpenSend">向新打开的窗口传递</button>
        <button id = "btnPreloadSend">向预加载的窗口传递</button>
    </div>
    <script src = "js/mui.min.js"></script>
    <script>
        //初始化预加载页面 new02.html
        mui.init({
            preloadPages: [{
                id: 'new02.html',
                url: 'new02.html'
            }]
        });
        document.getElementById('btnOpenSend')
            .addEventListener('tap', function() {
                //打开新页面 new01.html 并传值
                mui.openWindow({
                    url: "new01.html",
                    id: "new01.html",
                    extras: {
                        mydata: "1.hello HTML5"
                    }
                });
            });
        mui.plusReady(function() {
            document.getElementById('btnPreloadSend')
                .addEventListener('tap', function() {
                    var view = plus.webview.getWebviewById("new02.html");
                    if(view) {
                        //执行 new02.html 页面上的 GetData 事件,并传值

```

如图 13-5 所示效果, index.html 的作用就是显示绑定导航, new01.html 显示传值内容


```

        mui.fire(view, "GetData", {
            mydata: "2.hello HTML5"
        });
        view.show();
    }
    });
</script>
</body>
</html>

```

(3) 打开 new01.html, 页面代码修改如下:

```

<body>
  <header class="mui-bar mui-bar-nav">
    <a class="mui-action-back mui-icon
      mui-icon-left-nav mui-pull-left"></a>
    <h1 class="mui-title">新打开的页面</h1>
  </header>
  <script src="js/mui.min.js"></script>
  <script type="text/javascript">
    mui.init();
    mui.plusReady(function(){
      var self = plus.webview.currentWebview();
      var data = self.mydata;
      if(data){
        alert("取到值了, 数据为: " + data);
      }
    });
  </script>
</body>

```

(4) 打开 new02.html, 页面代码修改如下:

```

<body>
  <header class="mui-bar mui-bar-nav">
    <a class="mui-action-back
      mui-icon mui-icon-left-nav mui-pull-left"></a>
    <h1 class="mui-title">预加载的页面</h1>
  </header>
  <script src="js/mui.min.js"></script>
  <script type="text/javascript">
    mui.init();
    window.addEventListener("GetData", function(e){
      if(e.detail.mydata){
        alert("取到值了, 数据为: " + e.detail.mydata);
      }
    });
  </script>
</body>

```

真机运行后效果如图 13-8 所示,程序在主窗口中使用 openWindow 方法打开“new01. html”,对“new02. html”在 mui. init 方法中实现了预加载,两种方式都取到了主窗口传递过去的的数据。从这个例子可以看出,如果是打开新窗口同时是利用 openWindow 方法中的 extras 参数进行的数据传递,如果是已预加载的窗口,则利用在该窗口中实现自定义事件,在主窗口中使用 mui. fire 方法进行自动触发传递数据。

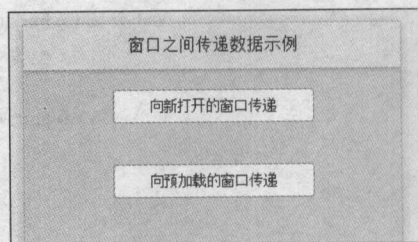


图 13-8 创建子页面效果

13.8 各种 UI 组件

13.8.1 按钮

1. 普通按钮

MUI 中的按钮使用的是< button >标签,在标签的 class 属性中使用“mui-btn”就可以生成默认按钮,默认是白色的,如果需要其他颜色的按钮,则继续增加对应的 class 属性值,MUI 中提供了蓝色(blue)、绿色(green)、黄色(yellow)、红色(red)、紫色(purple)五种色系的按钮,五种色系对应五种场景,分别为 primary、success、warning、danger、royal,要生成其他颜色的按钮,只需要在< button >标签的 class 属性中添加“mui-btn-场景值”,下面是生成各种颜色的标签按钮的 HTML 代码示范,运行效果如图 13-9 所示。

```
<button type="button" class="mui-btn">默认</button>
<button type="button" class="mui-btn mui-btn-primary">蓝色</button>
<button type="button" class="mui-btn mui-btn-success">绿色</button>
<button type="button" class="mui-btn mui-btn-warning">黄色</button>
<button type="button" class="mui-btn mui-btn-danger">红色</button>
<button type="button" class="mui-btn mui-btn-royal">紫色</button>
```

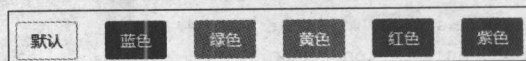


图 13-9 普通有底色按钮效果

如果希望实现无底色、有边框的按钮,并且底色使用父容器的背景色,就在 class 属性中再添加值“mui-btn-outlined”,例如制作实现一个蓝色边框无底色按钮,它的 HTML 代码应该书写为:

```
<button type="button"
class="mui-btn mui-btn-primary mui-btn-outlined">蓝色</button>
```

2. 带图标的按钮

有时为了让按钮看起来更漂亮一些,会在按钮中添加一些图标,MUI 使用了灵活的

icon 字体图标定义方式,在 MUI 按钮的 class 属性中添加“mui-icon mui-icon-图标 name 值”。图标的样式默认是居左的,如果希望居右,则在 class 属性中添加“mui-right”值。见图 13-10。下面是图标按钮的示范 HTML 代码:

```
<button type="button"
    class="mui-btn mui-btn-primary mui-icon mui-icon-home">首页
</button>
<button type="button"
    class="mui-btn mui-btn-primary mui-icon mui-icon-royal mui-right">
    查找
</button>
```

3. 带数字的按钮

在某些场景下,有些按钮需要显示数字,例如表示有多少条消息需要查阅,效果如图 13-11 所示,这时使用按钮结合数字角标做出漂亮的效果,但是要注意角标的场景风格应该与按钮的场景风格统一,这样做出的按钮的颜色效果显示起来才看起来协调,HTML 代码结构如下:

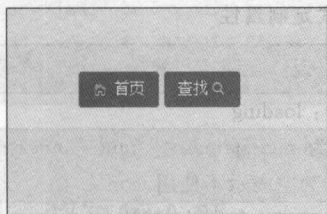


图 13-10 带图标的按钮

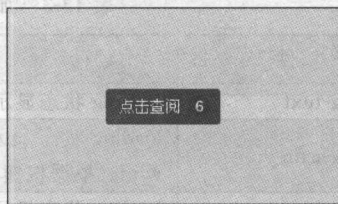


图 13-11 带数字的按钮

```
<button type="button" class="mui-btn mui-btn-royal">点击查阅
    <span class="mui-badge mui-badge-royal">6</span>
</button>
```

4. 块级按钮

在移动 App 应用中,块级按钮的运用是比较多的,也就是俗称的“大按钮”,MUI 中设计的块级按钮基本占满了屏幕的宽度,和屏幕的边框又保持了一定的边距,并且能自适应屏幕的宽度,自动扩大或缩小按钮的宽度,如图 13-12 所示。

在制作 MUI 普通按钮的基础上,要使按钮变成块级,需要在 class 属性列表中添加“mui-btn-block”,它的 HTML 代码结构如下:

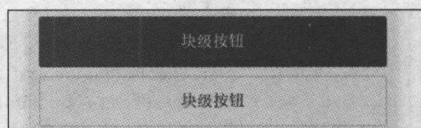


图 13-12 块级按钮

```
<button type="button" class="mui-btn mui-btn-royal mui-btn-block">
    块级按钮
</button>
<button type="button" class="mui-btn mui-btn-block mui-btn-outlined">
```



```
块级按钮
</button>
```

5. 加载中按钮

在很多情况下,当用户单击按钮后,需要向服务器端提交数据或等待服务器端响应时,常常需要提示类似于“正在提交”的文字,并将按钮设置为 disabled,用于避免用户重复单击,MUI 的加载中按钮就是用于实现类似的效果,这种效果和前面的按钮做法没有任何差别,主要是对相应的按钮使用两个 JavaScript API:

```
mui(btnElem).button('loading'); //切换为 loading 状态
mui(btnElem).button('reset'); //切换为 reset 状态(即重置为原始的 button)
```

对 MUI 中的按钮切换到 loading 状态后,按钮将不能再单击,同时按钮上文字会变成“Loading...”,文字左侧也会出现一个进度条。要想改变加载的一些样式,可以借助于在 < button > 标签中使用以下 3 个属性进行自定义,如表 13-3 所示。

表 13-3 加载中的按钮样式定制属性

属 性	说 明
data-loading-text	loading 状态显示的文字,默认为: loading
data-loading-icon	loading 状态显示的 icon,默认为 mui-spinner 或 mui-spinner mui-spinner-white(根据按钮样式自动识别),为空表示不使用 icon
data-loading-icon-position	loading 状态显示的 icon 的位置,支持 left/right 默认 left

以下是使用的 HTML 结构示例,运行结果如图 13-13 所示。

```
<div>
  <button type="button" class="mui-btn mui-btn-primary">确认</button>
  <button type="button" class="mui-btn mui-btn-primary"
    data-loading-icon-position="right"
    data-loading-text="请稍候...">确认</button>
</div>
<script>
  mui(document).on('tap', 'button', function(e) {
    mui(this).button('loading');
    setTimeout(function() {
      mui(this).button('reset');
    }.bind(this), 12000);
  });
</script>
```



图 13-13 加载中按钮

13.8.2 数字角标

数字角标一般和其他控件(列表、九宫格、选项卡等)配合使用,用于进行数量提示。角标的核心类是 `.mui-badge`,默认为实心灰色背景;同时,mui 还内置了蓝色(blue)、绿色(green)、黄色(yellow)、红色(red)、紫色(purple)五种色系的数字角标,示例代码如下:

```
<span class="mui-badge">1</span>
<span class="mui-badge mui-badge-primary">12</span>
<span class="mui-badge mui-badge-success">123</span>
<span class="mui-badge mui-badge-warning">3</span>
<span class="mui-badge mui-badge-danger">45</span>
<span class="mui-badge mui-badge-purple">456</span>
```

如果不需要底色,则增加 `.mui-badge-inverted` 类即可,如下:

```
<span class="mui-badge mui-badge-inverted">1</span>
<span class="mui-badge mui-badge-primary mui-badge-inverted">2</span>
<span class="mui-badge mui-badge-success mui-badge-inverted">3</span>
<span class="mui-badge mui-badge-warning mui-badge-inverted">4</span>
<span class="mui-badge mui-badge-danger mui-badge-inverted">5</span>
<span class="mui-badge mui-badge-royal mui-badge-inverted">6</span>
```

运行后显示效果如图 13-14 所示。

13.8.3 数字输入框

数字输入框是移动 App 中常见的 UI 组件(例如商品的数量),MUI 提供了数字输入框控件,可直接输入数字,也可以单击旁边的“+”、“-”按钮变换当前数值;默认的数字输入框 UI 组件结构比较简单,数字的读取和设置和普通输入框没有区别,示例代码如下:

```
<div class="mui-numbox">
  <!-- "-"按钮,单击可减小当前数值 -->
  <button class="mui-btn mui-numbox-btn-minus" type="button">-</button>
  <input class="mui-numbox-input" type="number" />
  <!-- "+"按钮,单击可增大当前数值 -->
  <button class="mui-btn mui-numbox-btn-plus" type="button">+</button>
</div>
```

运行效果如图 13-15 所示,数字输入框组件为了提供数字输入控制的灵活性,特别定义了以下几个属性设置输入框的参数,如表 13-4 所示。



图 13-14 数字角标

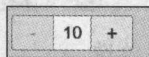


图 13-15 数字输入框

表 13-4 数字输入框的自定义属性

属 性	说 明
data-numbox-min	输入框允许使用的最小值,默认无限制
data-numbox-max	输入框允许使用的最大值,默认无限制
data-numbox-step	每次单击“+”、“-”按钮变化的步长,默认步长为 1

13.8.4 列表

1. 普通列表

列表是常用的 UI 控件,MUI 封装的列表组件比较简单,只需要在 ul 节点上添加 .mui-table-view 类、在 li 节点上添加 .mui-table-view-cell 类即可。示例代码如下:

```
<ul class = "mui-table-view">
  <li class = "mui-table-view-cell"> Item 1 </li>
  <li class = "mui-table-view-cell"> Item 2 </li>
  <li class = "mui-table-view-cell"> Item 3 </li>
</ul>
```

运行效果如图 13-16 所示。

若右侧需要增加导航箭头,变成导航链接,则只需在 li 节点下增加 a 子节点,并为该 a 节点增加 .mui-navigate-right 类即可,代码如下,运行效果如图 13-17 所示。

```
<li class = "mui-table-view-cell">
  <a class = "mui-navigate-right"> Item 1 </a>
</li>
```

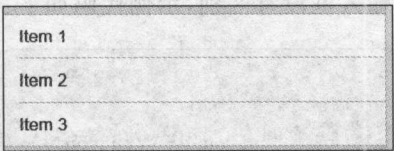


图 13-16 普通列表

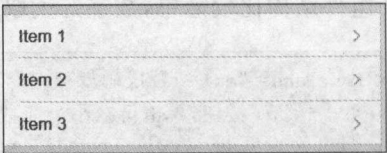


图 13-17 右侧带箭头的普通列表



单击列表, 对应列表项显示灰色高亮, 若想自定义高亮颜色, 只需要重新定义一个样式类 “.mui-table-view-cell.mui-active” 进行覆盖即可, 如下:

```
/* 单击变蓝色高亮 */
.mui-table-view-cell.mui-active{
  background-color: #0062CC;
}
```

2. 列表式单选

如果需要对列表实现单选,如图 13-18 所示,只需要在列表根节点上增加 .mui-table-

view-radio 类即可,若要默认选中某项,只需要在对应 li 节点上增加 .mui-selected 类即可,HTML 代码如下:

```
<ul class="mui-table-view mui-table-view-radio">
  <li class="mui-table-view-cell">
    <a class="mui-navigate-right">Item 1</a>
  </li>
  <li class="mui-table-view-cell mui-selected">
    <a class="mui-navigate-right">Item 2</a>
  </li>
  <li class="mui-table-view-cell">
    <a class="mui-navigate-right">Item 3</a>
  </li>
</ul>
```

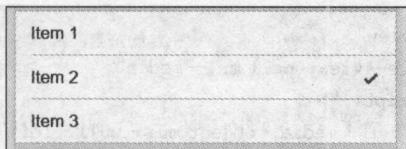


图 13-18 列表式单选效果

3. 滑动触发列表项

在列表项中,如果要实现类似 iOS 的短信列表项左滑删除效果,需要对普通列表项 li 改变如下:

```
<li class="mui-table-view-cell">
  <div class="mui-slider-right mui-disabled">
    <a class="mui-btn mui-btn-red">删除</a>
  </div>
  <div class="mui-slider-handle">
    要删除的列表项
  </div>
</li>
```

把上面的代码中的“mui-slider-right”修改成“mui-slider-left”就可以实现右滑动,运行后的效果如图 13-19 所示。

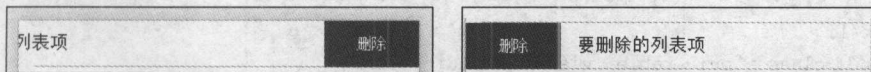


图 13-19 左右滑动显示删除按钮

如果希望左右滑动都能显示删除按钮,则可以对 li 改变如下:

```
<li class="mui-table-view-cell">
  <div class="mui-slider-left mui-disabled">
    <a class="mui-btn mui-btn-red">删除</a>
```

```

</div>
<div class="mui-slider-right mui-disabled">
  <a class="mui-btn mui-btn-red">删除</a>
</div>
<div class="mui-slider-handle">
  要删除的列表项
</div>
</li>

```

4. 图文列表

图文列表基本上成了移动 App 标准显示方式, MUI 中的图文列表组件继承自列表组件, 主要添加了 .mui-media、.mui-media-object、.mui-media-body、.mui-pull-left/right 几个类, 如下为示例代码:

```

<ul class="mui-table-view">
  <li class="mui-table-view-cell mui-media">
    <a href="javascript:;">
      
      <div class="mui-media-body">
        幸福
        <p class="mui-ellipsis">
          能和心爱的人一起睡觉, 是件幸福的事情; 可是, 打呼噜怎么办?
        </p>
      </div>
    </a>
  </li>
  <li class="mui-table-view-cell mui-media">
    <a href="javascript:;">
      
      <div class="mui-media-body">
        木屋
        <p class="mui-ellipsis">
          想要这样一间小木屋, 夏天刨冰吃瓜, 冬天围炉取暖。
        </p>
      </div>
    </a>
  </li>
  <li class="mui-table-view-cell mui-media">
    <a href="javascript:;">
      
      <div class="mui-media-body">
        CBD
        <p class="mui-ellipsis">
          烤炉模式的城, 到黄昏, 如同打翻的调色盘一般。
        </p>
      </div>
    </a>
  </li>

```

```

        </p>
      </div>
    </a>
  </li>
</ul>

```

缩略图片的位置可以自行控制居左或居右,主要是设定标签的 class 属性使用 .mui-pull-left 或者 .mui-pull-right 类,运行效果如图 13-20 所示。

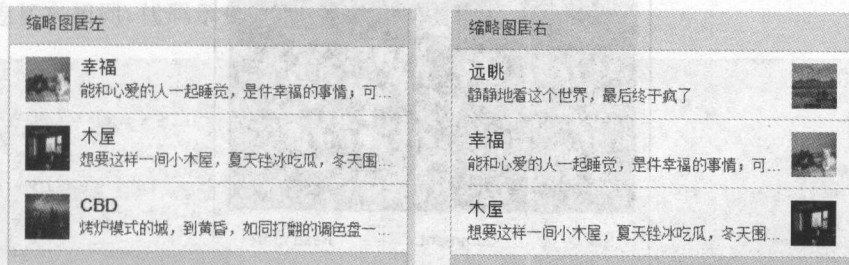


图 13-20 图文列表显示效果

13.8.5 折叠面板

折叠面板常用于功能分组显示或隐藏,MUI 中的折叠面板组件代码如下:

```

<ul class="mui-table-view">
  <li class="mui-table-view-cell mui-collapse">
    <a class="mui-navigate-right" href="#">面板 1</a>
    <div class="mui-collapse-content">
      <p>面板 1 子内容</p>
    </div>
  </li>
</ul>

```

可以在折叠面板中放置任何内容;折叠面板默认收缩,若希望某个面板默认展开,只需要在包含 .mui-collapse 类的 li 节点上增加 .mui-active 类即可。折叠面板的运行效果如图 13-21 所示:

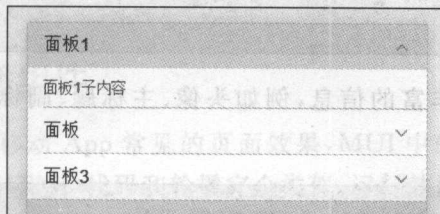


图 13-21 折叠面板显示效果

13.8.6 卡片视图

卡片视图常用于展现一段完整独立的信息,例如一篇文章的预览图、作者信息、点赞数量等,如图 13-22 所示是一个卡片 demo 示例。



图 13-22 卡片视图显示效果

使用 mui-card 类即可生成一个卡片容器,卡片视图主要有页眉、内容区、页脚三部分组成,结构如下:

```
<div class="mui-card">
  <!-- 页眉,放置标题 -->
  <div class="mui-card-header">页眉</div>
  <!-- 内容区 -->
  <div class="mui-card-content">内容区</div>
  <!-- 页脚,放置补充信息或支持的操作 -->
  <div class="mui-card-footer">页脚</div>
</div>
```

卡片页眉及内容区,均支持放置图片;如果在页眉放置图片,需要在 .mui-card-header 节点上增加 .mui-card-media 类,然后设置一张图片做背景图即可,代码示例如下:

```
<div class="mui-card-header mui-card-media"
  style="height:40vw;background-image:url(../images/cbd.jpg)">
</div>
```

若希望在页眉放置更丰富的信息,例如头像、主标题、副标题,则需使用 .mui-media-body 类,示例代码如下:

```
<div class="mui-card-header mui-card-media">
  
  <div class="mui-media-body">
    小 M
```

```
<p>发表于 2016 - 06 - 30 15:30 </p>
</div>
</div>
```

13.8.7 轮播组件

轮播组件是 MUI 提供的一个核心组件,在该核心组件基础上,衍生出了图片轮播、可拖动式图文表格、可拖动式选项卡、左右滑动九宫格等组件,这些组件有较多共同点。首先,HTML 构造基本相同,代码示例如下:

```
<div class = "mui-slider">
  <div class = "mui-slider-group">
    <!-- 第一个内容区容器 -->
    <div class = "mui-slider-item">
      <!-- 具体内容 -->
    </div>
    <!-- 第二个内容区 -->
    <div class = "mui-slider-item">
      <!-- 具体内容 -->
    </div>
  </div>
</div>
```

当拖动切换显示内容时,会触发 slide 事件,通过该事件的事件对象参数对象的 detail.slideNumber 属性可以获得当前显示项的索引(从 0 开始的),利用该事件,可在显示内容切换时,动态处理一些业务逻辑,示例代码如下:

```
document.querySelector('.mui-slider').addEventListener('slide', function(event) {
  if (event.detail.slideNumber == 1) {
    //切换到第二个选项卡
    //根据具体业务,动态获得第二个选项卡内容;
  } else if (event.detail.slideNumber == 2) {
    //切换到第三个选项卡
    //根据具体业务,动态获得第三个选项卡内容;
  }
});
```

13.8.8 图片轮播组件

图片轮播组件是网页和移动 App 常见的页面效果,MUI 中的图片轮播组件是从轮播组件继承下来的,所以它的 HTML 代码和轮播完全类似,运行效果如图 13-23 所示,代码示范如下:

```
<div id = "slider" class = "mui-slider">
```

```

<div class="mui-slider-group">
  <!-- 第一张 -->
  <div class="mui-slider-item">
    <a href="#">
      
    </a>
  </div>
  <!-- 第二张 -->
  <div class="mui-slider-item">
    <a href="#">
      
    </a>
  </div>
  <!-- 第三张 -->
  <div class="mui-slider-item">
    <a href="#">
      
    </a>
  </div>
  <!-- 第四张 -->
  <div class="mui-slider-item">
    <a href="#">
      
    </a>
  </div>
</div>
<!-- 图片移动标记圆点 -->
<div class="mui-slider-indicator">
  <div class="mui-indicator mui-active"></div>
  <div class="mui-indicator"></div>
  <div class="mui-indicator"></div>
  <div class="mui-indicator"></div>
</div>
</div>

```

示范中使用了4张图片,从第1张图片起,可以依次向左滑动进行图片切换,当切换到第4张图片时,继续向左滑动,无反应,继续显示第4张图片,用户若要显示第1张图片,必须反向右滑动,再切换回第1张图片。如果想让图片在第4张后左滑后继续出现第1张图片,需要在.mui-slider-group节点上增加.mui-slider-loop类,同时需要重复增加2张图片,图片顺序变为:4、1、2、3、4、1,代码修改示范如下:



图 13-23 图片轮播效果

```

<div id="slider" class="mui-slider">

```



```

<div class="mui-slider-group mui-slider-loop">
  <!-- 支持循环,需要重复图片节点 -->
  <div class="mui-slider-item mui-slider-item-duplicate">
    <a href="#"></a>
  </div>
  <div class="mui-slider-item">
    <a href="#"></a>
  </div>
  <div class="mui-slider-item">
    <a href="#"></a>
  </div>
  <div class="mui-slider-item">
    <a href="#"></a>
  </div>
  <div class="mui-slider-item">
    <a href="#"></a>
  </div>
  <!-- 支持循环,需要重复图片节点 -->
  <div class="mui-slider-item mui-slider-item-duplicate">
    <a href="#"></a>
  </div>
</div>
</div>

```

MUI 框架会默认初始化当前页面的图片轮播组件,但不会自动实现轮播,另外如果轮播组件图片为 JavaScript 动态生成时(例如通过 AJAX 动态获取的营销信息),则需要在动态生成完整 DOM (包含 mui-slider 下所有 DOM 结构)后,手动调用图片轮播的初始化方法,设定是否自动轮播及轮播周期,代码如下:

```

//获得 slider 插件对象
var gallery = mui('#slider');
gallery.slider({
  interval:5000//自动轮播周期,若为 0 则不自动播放,默认为 0;
});

```

如果需要程序跳转到第 x 张图片,则可以使用图片轮播插件的 gotoItem 方法,例如:

```

//获得 slider 插件对象
var gallery = mui('#slider');
gallery.slider().gotoItem(index); //跳转到第 index 张图片,index 从 0 开始;

```



对轮播图片组件中的图片高度的设置,需要自行设计 CSS 样式进行覆盖,例如:

```

.mui-slider-item img {height: 130px;}

```

13.8.9 复选框和单选框组件

MUI 框架中的复选框和单选框和 HTML 中的差异不大,checkbox 常用于多选的情况,例如批量删除、添加群聊等,radio 用于单选的情况。MUI 对它们作了封装。

复选框封装的 HTML 代码如下,默认 checkbox 在右侧显示,若希望在左侧显示,只需在 div 的 class 属性中增加 .mui-left 类即可,运行效果如图 13-24 所示。

```
<div class="mui-input-row mui-checkbox">
  <label>苹果</label>
  <input name="checkbox1" value="Apple" type="checkbox" checked>
</div>
```

单选框的封装和复选框基本类似,不同的是<input>标签的 type 属性和 div 所用的 class 属性,代码如下,运行效果如图 13-24 所示。

```
<div class="mui-input-row mui-radio">
  <label>苹果</label>
  <input name="gender" value="boy" type="radio" checked>
</div>
```



图 13-24 单选和复选效果

13.8.10 开关组件

MUI 提供了开关组件,单击和滑动这两种手势都可以对开关控件进行操作,UI 效果如图 13-25 所示。



图 13-25 开关组件效果

默认开关控件,带 ON/OFF 文字提示,打开时为绿色背景,基本的 class 类为 .mui-switch、.mui-switch-handle,HTML 代码结构如下:

```
<div class="mui-switch">
  <div class="mui-switch-handle"></div>
</div>
```

若希望开关默认为打开状态,只需要在 .mui-switch 节点上增加 .mui-active 类即可,例如:

```
<!-- 开关打开状态,多了一个.mui-active类 -->
<div class="mui-switch mui-active">
  <div class="mui-switch-handle"></div>
</div>
```

若希望隐藏 ON/OFF 文字提示,变成简洁模式,需要在 .mui-switch 节点上增加 .mui-switch-mini 类,如下:

```
<!-- 简洁模式开关关闭状态 -->
<div class="mui-switch mui-switch-mini">
  <div class="mui-switch-handle"></div>
</div>
<!-- 简洁模式开关打开状态 -->
<div class="mui-switch mui-switch-mini mui-active">
  <div class="mui-switch-handle"></div>
</div>
```

对于开关组件所封装的 API,可通过判断当前开关控件是否包含 .mui-active 类来实现,若包含,则为打开状态,否则即为关闭状态,如果需要用程序打开或关闭开关,可使用它的 toggle() 方法,如下为示例代码:

```
mui("#mySwitch").switch().toggle();
```

开关组件在打开/关闭两种状态之间进行切换时,会触发 toggle 事件,通过事件的 detail.isActive 属性可以判断当前开关状态。通过监听 toggle 事件,可以在开关切换时执行特定业务逻辑。如下为使用示例:

```
document.getElementById("mySwitch")
  .addEventListener("toggle",function(event){
    if(event.detail.isActive){
      console.log("开关启用");
    }else{
      console.log("开关关闭");
    }
  })
```

13.8.11 滑块组件

滑块常用于区间数字选择,MUI 中的滑块组件运行效果如图 13-26 所示,它的 HTML 结构如下:

```
<div class="mui-input-row mui-input-range">
  <label>滑块</label>
  <input type="range" min="0" max="100" id="rangeNumber">
</div>
```


滑块提供了一个 input 事件用于监听值的变化,可以通过 value 属性读取当前的值,示例代码如下:

```
document.getElementById('rangeNumber')
    .addEventListener('input',function () {
        console.log(this.value);
    });
```

13.8.12 字体图标组件

MUI 默认提供了手机 App 开发常用的字体图标,效果如图 13-27 所示,在使用它时,只需要在 span 节点上分别增加 .mui-icon、.mui-icon-name 两个类即可(name 为图标名称,例如: weixin、weibo 等),在 <http://dev.DCloud.net.cn/mui/ui/#icon> 页面上,列举了 MUI 自带的所有字体图标,每个图标下面都有已定义的样式类名,如下代码即可显示一个微信图标:

```
<span class = "mui-icon mui-icon-weixin"></span>
```

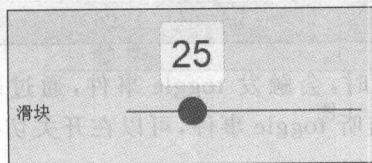


图 13-26 滑块组件效果



图 13-27 字体图标效果



要使用 MUI 自带的各种字体图标,必须使用 MUI 的字体文件“mui.ttf”。

13.8.13 表单组件

1. 表单结构

表单的样式设计在页面设计中相对比较麻烦,MUI 中为了简化表单输入框的设计,也对其进行了封装,例如下面的代码,所有包裹在“.mui-input-row”类中的 input、textarea 等元素都将被默认设置属性“width: 100%”。如果将 label 元素和上述控件包裹在“.mui-input-group”中可以获得最好的排列。运行后的效果如图 13-28 所示。

```
<form class = "mui-input-group">
    <div class = "mui-input-row">
        <label>用户名</label>
        <input type = "text" class = "mui-input-clear"
            placeholder = "请输入用户名">
```

```

</div>
<div class="mui-input-row">
  <label>密码</label>
  <input type="password" class="mui-input-password"
    placeholder="请输入密码">
</div>
<div class="mui-button-row">
  <button type="button" class="mui-btn mui-btn-primary">
    确认
  </button>
  <button type="button" class="mui-btn mui-btn-danger">
    取消
  </button>
</div>
</form>

```

2. 增强的 input 输入

- 只需要在<input>标签的 class 属性中添加. mui-input-clear 类, 当它中间有内容时, 右侧会有一个删除图标, 单击会清空当前<input>标签中的内容, 运行效果如图 13-29 所示, 示例代码如下:

```

<form class="mui-input-group">
  <div class="mui-input-row">
    <label>快速删除</label>
    <input type="text" class="mui-input-clear"
      placeholder="请输入内容">
  </div>
</form>

```

图 13-28 表单效果

图 13-29 有删除图标的输入框

- 在. mui-input-row 的 div 的样式中添加. mui-input-search 类, 就可以使用 type = "search" 的<input>标签, 运行效果见图 13-30, 有文字输入时, 图标会自动居左, 示例代码如下:

```

<div class="mui-input-row mui-search">
  <input type="search" class="mui-input-clear" placeholder="">
</div>

```

- 为了方便快速输入, MUI 集成了 HTML5+ 的语音输入 (采用了集成的科大讯飞公

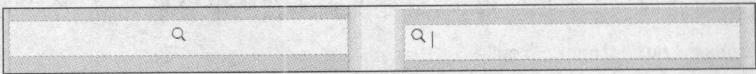


图 13-30 有搜索图标的输入框

司技术),只需要在对应<input>标签上添加 .mui-input-speech 类,就可以在 HTML5+ 环境下使用语音输入,运行效果如图 13-31 所示,单击图标后会自动打开语音输入界面,如图 13-32 所示。



图 13-31 有语音图标的输入框

- 对于密码,MUI 提供了专用的封装,只需要把类 .mui-input-password 给<input>标签加入 Class 属性,示例代码如下,运行效果如图 13-33 所示。



图 13-32 语音输入界面

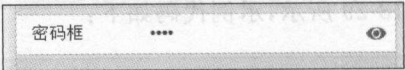



图 13-33 密码输入框

```
<div class = "mui - input - row">
  <label>密码框</label>
  <input type = "password" class = "mui - input - password"
    placeholder = "请输入密码">
</div>
```

 mui.init() 中会自动初始化基本各表单输入组件,但是动态添加的元素需要重新进行初始化,例如:

```
mui('.mui - input - row input').input();
```

13.8.14 进度条组件

在内容加载缓慢或页面处理数据的时候,使用进度条告诉用户处理进度是一种非常好的方式。MUI 中对于进度条的 HTML 封装代码如下。运行后,MUI 会自动在页面生成如图 13-34 所示的进度条。

```
<div id = "demo1" class = "mui - progressbar">
  <span></span>
</div>
```


- 若需要手工修改进度条的进度,则代码如下,运行效果如图 13-35 所示。

```
mui("#demo1").progressbar().setProgress(20); //20 是进度值
```

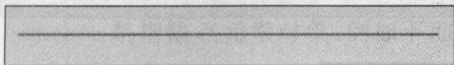


图 13-34 默认的进度条组件



图 13-35 进度 20% 的进度条组件

- 若需要动态创建进度条,示例如下:

```
mui(container).progressbar({progress:20}).show(); //20 是进度值
```

MUI 会检查当前容器自身是否包含 .mui-progressbar 类,若包含,则以当前容器为目标控件,直接显示进度,否则,检查当前容器的直接孩子节点是否包含 .mui-progressbar 类,若存在,则以遍历到的第一个含有 .mui-progressbar 类的孩子节点为目标控件显示进度,若当前容器的直接孩子节点,均不含 .mui-progressbar 类,则自动创建进度条控件。

- 若需要关闭进度条,则代码如下:

```
mui("#demo1").progressbar().hide();
```



关闭进度条一般用于动态创建(DOM 中预先未定义)的进度条,调用 hide 方法后,会直接删除对应的 DOM 节点;若已提前创建好 DOM 节点的进度条,调用 hide 方法无效。

- 若无法准确提供当前进度,可以提供无限循环进度条,无限循环进度条和准确值的进度条的用法基本相同,只是进度条控件 DOM 结构多了 .mui-progressbar-infinite,其他的完全一样,示例如下:

```
<div id="demo1" class="mui-progressbar mui-progressbar-infinite">
  <span></span>
</div>
```

- 页面顶部进度条类似于浏览器进度条,固定显示在页面顶部(标题导航控件下方);因此,若当前页面使用父子双 Webview 模式,子页面没有标题导航组件,则需通过自定义 css 的方式,重定义顶部进度条的位置,例如:

```
body>.mui-progressbar{
  top:0
}
```

使用页面顶部进度条时,无需编写 DOM 结构,使用如下代码即可自动创建,运行效果

如图 13-36 所示。

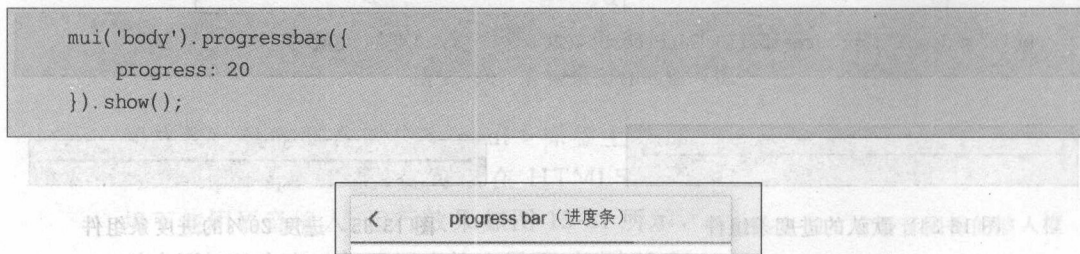


图 13-36 顶部进度条效果

13.8.15 弹出菜单组件

MUI 框架内置了弹出菜单插件,弹出菜单显示内容不限,但必须包裹在一个 class 属性包含“.mui-popover”类的 div 中,如下即为一个弹出菜单内容:

```
<div id="popover" class="mui-popover">
  <ul class="mui-table-view">
    <li class="mui-table-view-cell"><a href="#">Item1</a></li>
    <li class="mui-table-view-cell"><a href="#">Item2</a></li>
    <li class="mui-table-view-cell"><a href="#">Item3</a></li>
    <li class="mui-table-view-cell"><a href="#">Item4</a></li>
    <li class="mui-table-view-cell"><a href="#">Item5</a></li>
  </ul>
</div>
```

要显示、隐藏如上菜单,MUI 推荐使用锚点方式,例如:

```
<a href="#popover" id="openPopover"
  class="mui-btn mui-btn-primary mui-btn-block">打开弹出菜单
</a>
```

运行后效果如图 13-37 所示,单击定义的按钮,即可显示弹出菜单,并且在菜单下面会出现一个遮罩层;再次单击弹出菜单之外的其他区域,均可关闭弹出菜单和遮罩层。这种使用方式最为简洁。若希望通过 JS 的方式控制弹出菜单,则通过如下一个方法即可:

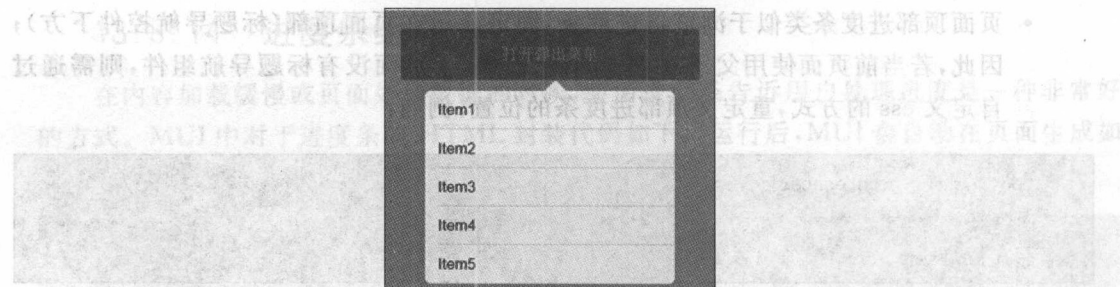


图 13-37 弹出菜单效果

```
mui('#popover').popover(status[, anchor]);
```

status 是个 String 类型,“show”显示 popover,“hide”隐藏 popover,“toggle”自动识别处理显示隐藏状态。

anchor 是指锚点元素对象,例如下面的代码:

```
//传入 toggle 参数,用户也无需关心当前是显示还是隐藏状态,mui 会自动识别处理;
mui('#popover').popover('toggle',
    document.getElementById("openPopover"));
```

13.8.16 遮罩层组件

在弹出菜单、侧滑菜单等界面,经常会用到遮罩层,例如弹出菜单弹出后,除菜单外的其他区域都会遮罩一层遮罩,用户单击遮罩不会触发遮罩下方的逻辑,而会在关闭弹出菜单的同时关闭遮罩层。

遮罩层常用的操作包括创建、显示、关闭,代码如下:

```
var mask = mui.createMask(callback);
//callback 为用户点击蒙版时自动执行的回调;
mask.show();    //显示遮罩
mask.close();   //关闭遮罩
```

默认的遮罩使用 .mui-backdrop 类定义(如下代码),若需自定义遮罩效果,只需覆盖定义 .mui-backdrop 即可:

```
.mui-backdrop {
    position: fixed;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
    z-index: 998;
    background-color: rgba(0,0,0,.3);
}
```

13.8.17 操作表组件

操作表是目前手机系统中常见的一个效果,一般从底部弹出,显示一系列可供用户选择的操作按钮。MUI 中的操作表组件是从弹出菜单组件的基础上演变而来,实际上就是一个固定从底部弹出的菜单,所以 HTML 代码结构以及控制方式和弹出菜单非常类似,只需要在包含 .mui-popover 类的节点上增加 .mui-popover-bottom、.mui-popover-action 类,代码如下:


```
<div id="sheet1" class="mui-popover
      mui-popover-bottom mui-popover-action">
    //自定义界面的实现
</div>
```

通过使用标签和各种 CSS 样式,很容易能实现类似于图 13-38 中左侧的操作表效果,但这种操作表不支持覆盖顶部状态栏,不支持跨 Webview 的遮罩,在有 map 等原生控件时,容易被遮挡。在 HTML5+ 规范中也定义了原生的操作表组件的调用形式,但它只能实现简单的列表方式,不支持自定义样式,但它支持覆盖顶部状态栏,支持跨 Webview 的遮罩,在有 map 等原生控件时,依然会显示在最顶层,不会被遮挡,如图 13-38 中右侧的效果。

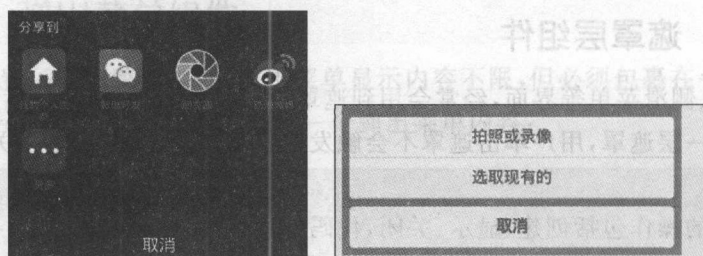


图 13-38 MUI 的操作表组件和 HTML5+ 中的原生操作表

13.8.18 对话框组件

HTML5+ 规范中提供了原生的一些对话框调用方式,主要是调用 plus.nativeUI 模块中的各方法。在 MUI 框架中也设计了一些相应的对话框,和 HTML5+ 中的不同的是,用户可以基于 CSS 样式做出不同风格的对话框,对话框有 4 种类型:提示对话框、确认对话框、输入对话框、自动消失提示框,这些提示框出现时,界面上都会出现遮罩层,图 13-39 是这 4 种对话框的效果,下面简单介绍这几种对话框的使用。

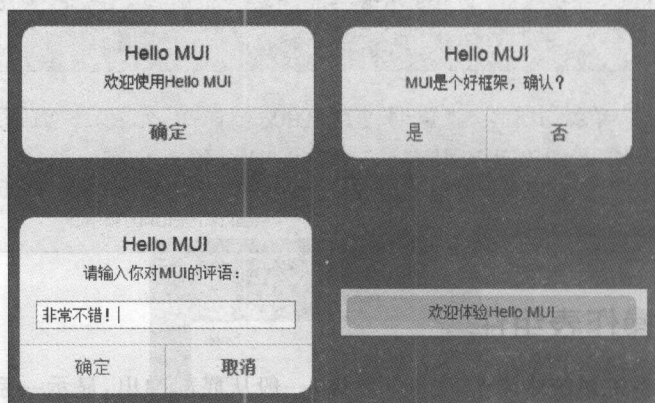


图 13-39 MUI 的对话框组件效果

1. 提示对话框

语法形式如下,各参数含义见表 13-5。

```
mui.alert(message, title, btnValue, callback[, type])
```

表 13-5 alert 方法各参数说明

属 性	说 明
message	String 类型,提示对话框上显示的内容
title	String 类型,提示对话框上显示的标题
btnValue	String 类型,提示对话框上按钮显示的内容
callback	提示对话框上关闭后的回调函数
type	可选参数,值为“div”时表示是否使用 h5 绘制的对话框

提示对话框的使用示例如下:

```
mui.alert('欢迎使用 Hello MUI', 'Hello MUI', function() {
    //回调处理代码
});
```

2. 确认对话框

语法形式如下,各参数含义见表 13-6:

```
mui.confirm(message, title, btnValue, callback[, type])
```

表 13-6 confirm 方法各参数说明

属性	说 明
message	String 类型,提示对话框上显示的内容
title	String 类型,提示对话框上显示的标题
btnValue	String 类型数组,提示对话框上按钮显示的内容
callback	提示对话框上关闭后的回调函数,通过事件参数对象的 index 属性来确定选择的是哪个按钮
type	可选参数,值为“div”时表示是否使用 h5 绘制的对话框

确认对话框的使用示例如下:

```
var btnArray = ['是', '否'];
mui.confirm('MUI 是个好框架,确认?', 'Hello MUI', btnArray,
    function(e) {
        if (e.index == 0) {
            info.innerText = '你刚确认 MUI 是个好框架';
        } else {
            info.innerText = 'MUI 没有得到你的认可,继续加油'
        }
    }
);
```

3. 输入对话框

语法形式如下,各参数含义见表 13-7。

```
mui.prompt(message,placeholder,title,btnValue,callback[,type])
```

表 13-7 prompt 方法各参数说明

属 性	说 明
message	String 类型,提示对话框上显示的内容
placeholder	String 类型,编辑框上显示的提示文字
title	String 类型,提示对话框上显示的标题
btnValue	String 类型数组,提示对话框上按钮显示的内容
callback	提示对话框上关闭后的回调函数,通过事件参数对象的 index 属性来确定选择的是哪个按钮
type	可选参数,值为“div”时表示是否使用 h5 绘制的对话框

输入对话框的使用示例如下：

```
var btnArray = ['确定', '取消'];
mui.prompt('请输入你对 MUI 的评语: ', '性能好', 'Hello MUI',
    btnArray, function(e) {
        if (e.index == 0) {
            console.log('谢谢你的评语: ' + e.value);
        } else {
            console.log('你单击了取消按钮!');
        }
    });
```

4. 自动消失对话框

语法形式如下：

```
mui.toast(message[,options]);
```

message 是 String 类型,表示消息框显示的文字内容。
options 是个 JSON 类型,表示提示消息的参数,各参数说明见表 13-8。

表 13-8 toast 方法消息参数说明

属性	说 明
duration	持续显示时间,默认值 short(2000ms),支持整数值和 String ,String 可选: long(3500ms), short(2000ms)
type	“div”,强制使用 mui 消息框(div 模式)

自动消失对话框的使用示例如下：

```
mui.toast('登录成功',{
  duration:'long',
  type:'div'
});
```

13.8.19 scroll 区域滚动

在 App 开发中,div 区域滚动的需求是普遍存在的,但系统默认实现又有如下问题:

- Android 平台 4.0 以下不支持 div 滚动;
- Android 平台 4.0 以上支持 div 滚动,但不显示滚动条;
- 弹出层的 div 滚动在 iOS 平台存在事件透传的问题。

因此,MUI 框架额外提供了区域滚动组件,使用时需要遵循如下 HTML 代码结构:

```
<div class="mui-scroll-wrApper">
  <div class="mui-scroll">
    <!-- 这里放置真实显示的 DOM 内容 -->
  </div>
</div>
```

区域滚动组件默认为 absolute 定位,全屏显示;在实际使用过程中,需要手动设置滚动区域的位置(top 和 height)。若要使用区域滚动组件,需手动初始化区域滚动组件,它的配置参数是个 JSON 对象,属性如下:

```
var options = {
  scrollY: true,      //是否竖向滚动
  scrollX: false,     //是否横向滚动
  startX: 0,         //初始化时滚动至 x
  startY: 0,         //初始化时滚动至 y
  indicators: true,  //是否显示滚动条
  deceleration: 0.0006 //阻尼系数,系数越小滑动越灵敏
  bounce: true,      //是否启用回弹
}
```

初始化时,只需要如下代码:

```
mui('.mui-scroll-wrApper').scroll(options);
```

MUI 框架还额外为区域滚动组件封装了部分方法。

- 滚动到特定位置,语法形式如下:

```
scrollTo(xpos,ypos[,duration]);
```

xpos 和 ypos 都是数值,表示要在窗口文档显示区左上角显示的文档的 x 坐标和 y 坐标,duraion 也是数值,表示滚动时间周期,单位是毫秒,使用示例如下:

```
mui('.mui-scroll-wrApper').scroll().scrollTo(0,0,100); //100 毫秒滚动到顶
```

- 滚动到底部位置,滚动到顶部的代码比较容易实现,坐标值设为 0,0 即可;但滚动到底部,需要计算该区域的实际高度,因此 MUI 封装了 scrollToBottom 方法,语法形式如下:

```
scrollToBottom(duraion); //duraion 是滚动时间周期,单位是毫秒
```

- 横向滚动,如图 13-40 所示,这是目前新闻客户端流行的横向滚动栏目,在 MUI 中只需在区域滚动组件基础上给 class 属性添加“mui-slider-indicatorcode、mui-segmented-control、mui-segmented-control-inverted”即可,示例代码如下:



图 13-40 横向滚动效果

```
<div class="mui-scroll">
  <a class="mui-control-item mui-active">
    头条
  </a>
  <a class="mui-control-item">
    要闻
  </a>
  <a class="mui-control-item">
    娱乐
  </a>
  <a class="mui-control-item">
    热点
  </a>
  <a class="mui-control-item">
    娱乐
  </a>
  <a class="mui-control-item">
    科技
  </a>
</div>
```

13.9 下拉刷新和上拉加载

为实现下拉刷新功能,大多 HTML5 框架都是通过 div 模拟下拉回弹动画,在低端 Android 手机上,div 的动画经常出现卡顿现象(特别是有图文列表的情况);MUI 框架通过

双 Webview 解决这个 div 的拖动流畅度问题；拖动时，拖动的不是 div，而是一个完整的 Webview(子 Webview)，回弹动画则使用原生动画；在 iOS 平台，HTML5 的动画已经比较流畅，故依然使用 HTML5 方案。两个平台实现虽有差异，但 MUI 框架经过封装，可使用一套代码实现下拉刷新。

13.9.1 下拉刷新

关于如何用 mui.init 方法生成子页在此不再赘述，请参看前面的讲解。这里主要讲解 Webview 中页面的设计，要实现刷新的页面的 HTML 代码结构如下所示：

```
<!-- 下拉刷新容器 -->
<div id="refreshContainer" class="mui-content mui-scroll-wrApper">
  <div class="mui-scroll">
    <!-- 数据列表 -->
    <ul class="mui-table-view mui-table-view-chevron">

      </ul>
    </div>
  </div>
```

其次，通过 mui.init 方法中 pullRefresh 参数配置下拉刷新各项参数，代码如下：

```
mui.init({
  pullRefresh: {
    container: "#refreshContainer", //下拉刷新容器 css 选择器标识
    down: {
      height: 50, //可选，默认 50. 触发下拉刷新拖动距离，
      auto: true, //可选，默认 false. 自动下拉刷新一次
      contentdown: "下拉可以刷新", //可选，在下拉可刷新状态时，下拉刷新控件上显示的标题
      contentover: "释放立即刷新", //可选，在释放可刷新状态时，下拉刷新控件上显示的标题
      contentrefresh: "正在刷新...", //可选，正在刷新状态时，下拉刷新控件上显示的标题内容
      callback: pulldownrefresh //必选，刷新函数，根据具体业务来编写，比如通过 ajax 从服务器获取新数据；
    }
  }
});
```

在下拉刷新过程中，当获取新数据后，需要执行 endPulldownToRefresh 方法，该方法的作用是关闭“正在刷新”的进度提示，内容区域回滚顶部位置，代码如下：

```
function pulldownrefresh() {
  //业务逻辑代码，比如通过 ajax 从服务器获取新数据；
  ...
  //注意，加载完新数据后，必须执行如下代码
```



```

mui('#refreshContainer').pullRefresh().endPulldownToRefresh();
}

```

当列表内容较长,需要迅速回滚到某个指定位置时,可以采用代码如下:

```

mui('#pullrefresh').pullRefresh().scrollTo(0,0,100)

```

13.9.2 上拉加载

MUI框架中的上拉加载和下拉刷新 HTML 结构完全类似,通过 mui.init 方法中 pullRefresh 参数配置上拉加载各项参数,代码如下:

```

mui.init({
  pullRefresh: {
    container: refreshContainer, //待刷新容器 css 选择器标识
    up: {
      height: 50, //可选,默认 50. 触发上拉加载拖动距离
      auto: true, //可选,默认 false. 自动上拉加载一次
      contentrefresh: "正在加载...", //可选,正在加载状态时,上拉加载控件上显示的标题内容
      contentnomore: "没有更多数据了", //可选,请求完毕若没有更多数据时显示的提醒内容;
      callback: pullupRefresh //必选,刷新函数,根据具体业务来编写,比如通过 ajax 从服务器
      获取新数据;
    }
  }
});

```

同样地,在上拉加载完新数据后,需要执行 endPullupToRefresh() 方法,结束转雪花进度条的“正在加载...”过程。不同的是,endPullupToRefresh 方法可以输入一个布尔值,用于标识是否还有更多数据。若还有更多数据,则传入 false;否则传入 true,之后滚动条滚动到底部时,将不再显示“上拉显示更多”的提示语,而显示“没有更多数据了”的提示语,示例代码如下:

```

function pullupRefresh () {
  //业务逻辑代码,比如通过 ajax 从服务器获取新数据;
  ...
  //注意:
  //1. 加载完新数据后,必须执行如下代码,true 表示没有更多数据了:
  this.endPullupToRefresh(true|false);
}

```

- 在部分业务中,有重新触发上拉加载的需求(例如当前类别已无更多数据,但切换到另外一个类别后,应支持继续上拉加载),此时调用 refresh(true) 方法,可重置上拉加载控件,代码如下:

```
mui('#pullup-container').pullRefresh().refresh(true);
```

- 在部分场景下希望禁用上拉加载,例如在列表数据过少时,不想显示“上拉显示更多”“没有更多数据”的提示语,开发者可以通过调用 `disablePullupToRefresh()` 方法实现类似需求,代码如下:

```
mui('#pullup-container').pullRefresh().disablePullupToRefresh();
```

- 使用 `disablePullupToRefresh()` 方法禁用上拉加载后,可通过 `enablePullupToRefresh()` 方法再次启用上拉加载,代码如下:

```
mui('#pullup-container').pullRefresh().enablePullupToRefresh();
```

13.10 MUI 的插件

为了简化开发者的多端发布开发,MUI 框架在核心库之外,补充了一些插件,这些插件不一定是和 UI 相关的,也有和业务相关的。在 MUI Demo 示例里下方的“模板”中的例子,基本都属于插件示例。这些插件的使用需要加载除 MUI 标准 JS 库之外的 JS 等资源。在 HBuilder 中建立好 MUI 的 Demo 项目后,它的 js 目录中就内置了很多插件的 JS 库。

限于本书篇幅,只简单介绍其中 4 个常用的插件。

13.10.1 延迟加载插件

所谓延迟加载,就是页面初始化时,暂不加载处于屏幕可见区域之外的图片,这样做的好处是可以加快页面渲染速度、提升页面滚动性能、默认不下载屏幕外的图片,减少流量。MUI 的延迟加载组件需要使用比较简单,先要准备好一张备用图片,用于在图片实现延迟加载时占位显示的图片,接着在页面引入 `mui.min.js` 后,再依次引入两个脚本,一个是“`mui.lazyload.js`”,另一个是“`mui.lazyload.img.js`”。

完成脚本引入后,在页面中加入以下代码:

```
$(document).imageLazyload({
  placeholder: '../images/60x60.gif' //这里是预先准备好的占位图片
});
```

这样,当页面的图文列表显示较长时,如果数据图片加载时间较长或在屏幕之外,会暂时先显示占位图片,然后在数据图加载完成或进入屏幕之内时再显示出来,效果如图 13-41 所示。

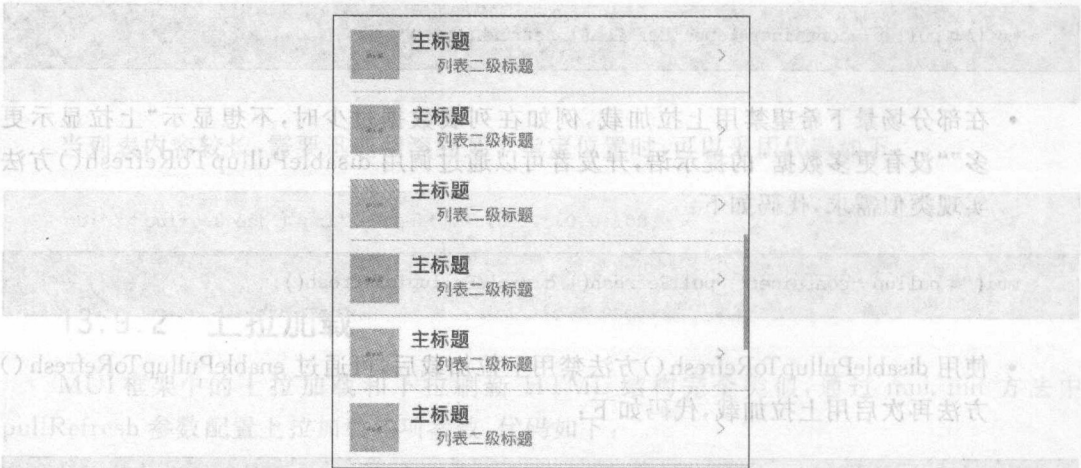


图 13-41 延迟加载效果

13.10.2 图片预览插件

如图 13-42 所示是图片预览效果,当页面上有很多图片时,使用图片预览插件后,单击图片就可以进入图片全屏预览功能。将图片全屏后,双击或双指缩放均可对图片进行放大、缩小操作,左右滑动可查看同组(data-preview-group 相同的图片为一组)其他图片,单击会关闭预览。



图 13-42 图片预览效果

这个插件使用时,首先是页面中的图片标签应该设计为如下代码:


```
<img src = "图片地址"
      data-preview-group = "分组号" data-preview-src = "预览的大图地址"/>
```

属性“data-preview-group”代表分组号,全屏预览中的图片都是同一组的,属性“data-preview-src”是全屏预览中要显示的图片路径,如果为空字符串,则和 src 的地址一致。

在页面引入“mui.min.js”后,再依次引入“mui.zoom.js”和“mui.previewimage.js”,另外,需要注意的是,这个插件还有自己配套的 CSS 样式设计,这部分内容较多,请自行参看 Demo 示例,在使用时必须同时复制相应的样式。

最后,这个插件还需要配置的代码,如下:

```
mui.previewImage();
```

属性“data-preview-src”必须保留,即使没有相应的大图显示,也要为空字符串。

13.10.3 日期和时间选择器插件

在多级选择器插件的基础上,MUI 框架封装了日期和时间,效果如图 13-43 所示。它依赖的 .js 文件和 .css 文件和多级选择器是一样的,不同的是它的语法形式,要生成日期和时间插件,先得生成这个对象,语法如下:

```
var dtPicker = new mui.DtPicker(options);
```



图 13-43 各种日期和时间选择效果

options 是配置的 JSON 对象,各配置属性如表 13-9 所示。

表 13-9 日期和时间配置属性

属 性	说 明
type	String 类型,'datetime'表示完整日期视图(年月日时分),'date'年视图(年月日),'time'时间视图(时分),'month'月视图(年月),'hour'时视图(年月日时)
customData	JSON 类型,设置时间/日期自定义数据,每个数据都以{text:文字显示,value:值}的 JSON 格式附加,'y'是修改年份的,'m'是月份,'d'是日期,'h'是小时,'m'是分钟
labels	数组对象,设置默认标签区域提示语,可设置["年","月","日","时","分"]这五个字段,可以根据视图模式选择设置个别标签,也可以设置所有标签,建议不要设置空字符串,会影响美观
beginDate	Date 对象,设置可选择日期最小范围
endDate	Date 对象,设置可选择日期最大范围

它显示后的回调函数使用如下:

```
dtpicker.show(function(items) {
    /* * items.value 拼合后的 value
    * items.text 拼合后的 text
    * items.y 年,可以通过 rs.y.vaue 和 rs.y.text 获取值和文本
    * items.m 月,用法同年
    * items.d 日,用法同年
    * items.h 时,用法同年
    * items.i 分(minutes 的第二个字母),用法同年
    * /
})
```

13.10.4 单页面刷新插件

MUI 框架中本身已经提供了下拉刷新和上拉加载功能,但它是基于 HTML5+ 的双 Webview 实现的下拉刷新效果,在普通浏览器转换成 iframe 实现同样的效果。整体页面刷新插件是仿照原生的 Material Design 风格的刷新插件,主要用于单页面效果,如图 13-44 所示。这个插件的使用除了核心的“mui.min.js”文件,还需要依赖其他两个.js 文件,需要依次引入“mui.pullToRefresh.js”和“mui.pullToRefresh.material.js”。

这个插件的下拉刷新进度条主要是利用了 canvas 的绘图,它的 HTML 代码结构如下:

```
<div id="slider" class="mui-slider mui-fullscreen">
  <div class="mui-scroll" id="pulls">
    <ul class="mui-table-view">
      //...数据的 li 列表项
    </ul>
  </div>
</div>
```



图 13-44 单页面刷新效果

插件的调用代码形式为：

```
mui("#pulls").pullToRefresh({
  down: {
    callback: function() {
      /* 刷新处理逻辑 */
      this.endPulldownToRefresh();
    }
  },
  up: {
    callback: function() {
      /* 刷新处理逻辑 */
      // 默认为 false, true 表没有数据了
      this.endPullupToRefresh(true|false);
    }
  }
});
```

这个插件的使用还有配套的 CSS 样式,代码如下:

```
.mui-pull-top-tips {
  position: absolute;
  top: -20px;
  left: 50%;
  margin-left: -25px;
  width: 40px;
  height: 40px;
  border-radius: 100%;
  z-index: 1;
}

.mui-bar~.mui-pull-top-tips {
  top: 24px;
}
```



```
.mui-pull-top-tips.mui-transitioning {
    -webkit-transition-duration: 200ms;
    transition-duration: 200ms;
}

.mui-pull-bottom-tips {
    text-align: center;
    background-color: #efeff4;
    font-size: 15px;
    line-height: 40px;
    color: #777;
}

.mui-pull-top-canvas {
    overflow: hidden;
    background-color: #fafafa;
    border-radius: 40px;
    box-shadow: 0 4px 10px #bbb;
    width: 40px;
    height: 40px;
    margin: 0 auto;
}

.mui-pull-top-canvas canvas {
    width: 40px;
}
```

13.11 MUI 的 AJAX 封装

MUI 框架基于 HTML5+ 中的 XMLHttpRequest，封装了常用的 AJAX 通信函数，它借鉴了 jQuery 中的 AJAX 通信的 API 设计，所提供的方法也和 jQuery 的基本类似。不同的是：jQuery 用的是 jQuery 对象，MUI 框架用的是 mui 对象。本着极简的设计原则，mui 也提供了 mui.ajax 方法，它的语法形式如下：

```
mui.ajax(settings);
```

settings 是 JSON 配置对象，它的各属性如表 13-10 所示。

表 13-10 setting 配置属性

属 性	说 明
type	String 类型，HTTP 请求的方法
url	String 类型，HTTP 请求的 url
data	String 类型或 JSON 数据
dataType	String 类型，预期服务器返回的数据类型；如果不指定，mui 将自动根据 HTTP 包的 MIME 头信息自动判断；支持设置的 dataType 可选值：“xml”、“html”、“script”、“json”、“text”

续表

属 性	说 明
crossDomain	Boolean 类型,强制使用 HTML5+跨域
error	请求失败时触发的回调函数,该函数接收三个参数:第一个参数是 XMLHttpRequest 对象;第二个参数是错误描述,包括 "timeout", "error", "abort", "parsererror", "null";第三个参数是可捕获的异常对象
success	请求成功时触发的回调函数,该函数接收三个参数:第一个参数是服务器返回的响应数据,类型可以是 json 对象、xml 对象、字符串等;第二个参数状态描述,默认值为 success;第三个参数是 XMLHttpRequest 对象
timeout	Number 类型,请求超时时间(毫秒),默认值为 0,表示永不超时;若超过设置的超时时间,则触发 error 回调
headers	JSON 类型,指定 HTTP 请求的 Header

在 mui.ajax 方法基础上,MUI 框架进一步简化出最常用的 mui.get()、mui.getJSON()、mui.post()三个方法,它们的语法形式如下:

```
//直接使用 GET 请求方式向服务器发送数据且不处理 timeout 和异常
mui.get(url[,data][,success][,dataType]);

//使用 POST 请求方式向服务器发送数据且不处理 timeout 和异常
mui.post(url[,data][,success][,dataType]);

//限定返回 json 格式的数据
mui.getJSON(url[,data][,success]);
```

13.12 Chrome 调试 Android 应用

调试是软件开发过程中很重要的环节,它能帮助开发者快速地定位和解决开发过程中碰到的问题。

在实际开发过程中,对于苹果设备的调试,一种是使用 Xcode 的 iOS 模拟器,这要求必须有 Mac 环境,另一种是使用 iPhone、iPad、iTouch 等真实设备。由于 iOS 有签名校验机制,还需要开发者账号,申请苹果开发证书(Certificates)和描述文件(Provisioning Profiles),相对比较麻烦,所以这里只介绍使用 Chrome 调试 Android 应用。

对于移动平台的开发者来说,从 Android4.4 开始,也可以通过 Chrome 的 DevTools 工具连接设备对应用进行调试。对于 HTML5 App 的开发,Chrome(要求版本为 30.0 以上)的 DevTools 工具有强大的功能和友好的用户体验,不仅能快速方便调试 JavaScript、检查 HTML 页面 DOM 结构、实时同步更新元素 CSS 样式,还能跟踪分析页面资源加载性能等问题。使用的步骤如下:

① 在 HBuilder 中新建“移动 App”项目后,编写完相应的 App 代码后,使用真机或模拟器运行相应的 App。这里以真机为例,通过“运行”菜单中的“手机运行”选择相应的设备运

行,也可以通过工具栏中的图标运行,如图 13-45 所示。需要注意的是:一定要保证手机的操作系统是 Android 4.4 以上,并且手机设备开启了 USB 调试功能。

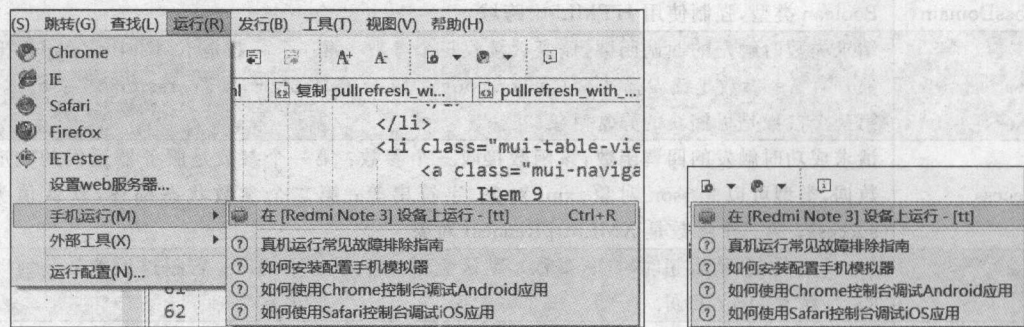


图 13-45 真机运行选择

② 在 Chrome 地址栏,输入“Chrome://inspect”命令,DevTools 工具会自动检测已连接设备运行的可调试页面列表,包括已经预加载的页面,如图 13-46 所示。

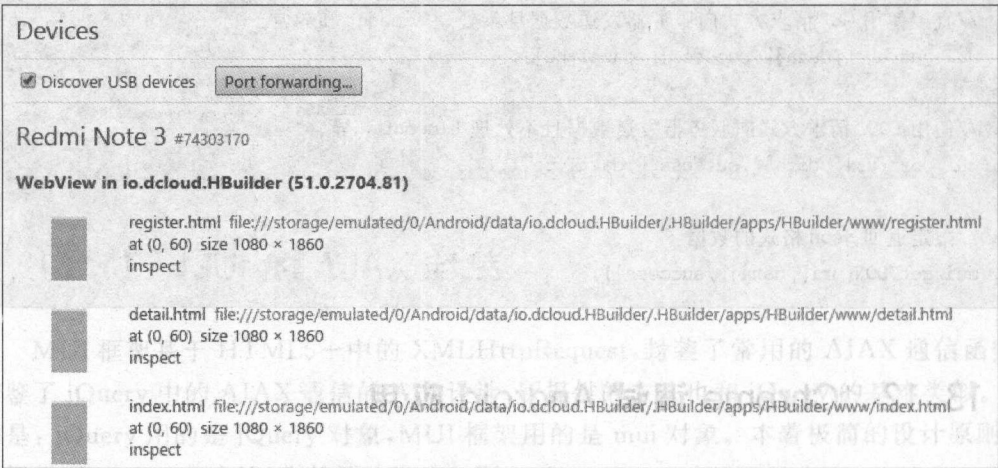


图 13-46 可调试页面列表

③ 选择要调试的页面,单击它下面对应的“inspect”链接,如果遇到启动了一个白屏界面,说明网站被屏蔽了,因为此服务需要连接 google 的服务器。开发者可以使用三方翻墙服务(在百度中搜索关键词“vpn”)。一般情况下,只有第一次使用“inspect”时需要翻墙,以后会缓存在本地。

④ 打开要调试页面后,DevTools 会自动加载相关资源,调试页面操作方式与普通 html 页面的调试完全一致。例如,在“Elements”下查看 DOM 结构,选中 DOM 元素后,在设备上会高亮显示,右侧 Styles 下修改 css 属性可即时生效,如图 13-47 所示。

⑤ 对于 App 中 JavaScript 的调试和普通页面调试没有区别,在“Sources”中找到源代码,设置好相应的断点,按 F5 键重新加载后进行断点调试,如图 13-48 所示。



图 13-47 DevTools 页面调试

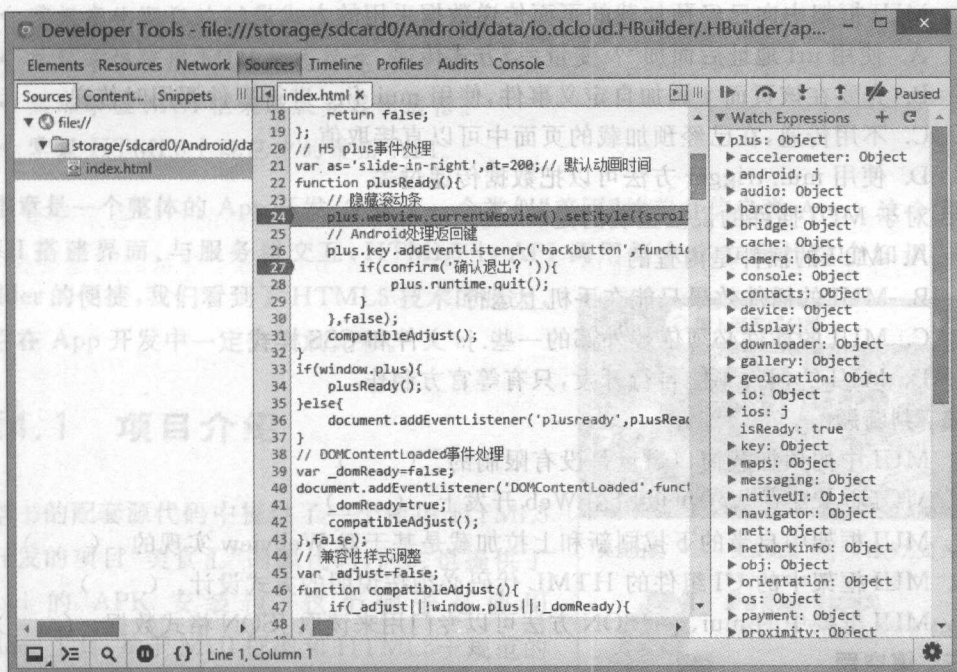


图 13-48 JavaScript 断点调试

小结

本章主要讲解了 DCloud 公司提供的高性能 App 开发框架——MUI。首先介绍了 MUI 框架,以及如何创建基于 MUI 的移动 App 项目;讲解了 MUI 项目中页面布局的一些特殊使用、MUI 中的事件管理、窗口管理,并详细讲解了 MUI 框架中提供的各种 UI 使用的要点;介绍了 MUI 框架中有特色的一些插件,MUI 中实现与服务器端 AJAX 通信的各方法,最后介绍了如何使用 Chrome 调试 Android 程序。本章的内容可灵活应用在 HTML5 手机 App 或 Web App 开发中,快速实现 App 窗口或 Web 页面的设计。

习题

一、选择题

- MUI 框架没有提供以下()功能。
 - UI 组件
 - 查找页面 HTML 元素对象
 - 各种 DOM 操作
 - AJAX 通信
- MUI 中页面使用自定义事件是使用()对象。
 - document
 - window
 - mui
 - body
- 在 MUI 框架中,创建子页面、关闭页面、手势事件配置、预加载等功能设置是利用 mui 对象的()方法实现的。
 - AJAX
 - init
 - preload
 - trigger
- MUI 框架中向已经预加载的页面传递数据采用的方式是()。
 - 使用 url 地址后面加“? 变量=”方式传递
 - 先要在该页面上添加自定义事件,使用 mui.fire 进行事件调用时传递
 - 不用传递,在已经预加载的页面中可以直接取值
 - 使用 mui.trigger 方法可以把数据传递过去
- 对于 MUI 插件的说法正确的是()。
 - MUI 的插件是内置的
 - MUI 的插件效果只能在手机上运行
 - MUI 的插件必须依赖外部的一些 .js 文件和 CSS 样式
 - MUI 的插件不能自行开发,只有等官方制作

二、判断题

- MUI 中的预加载窗口数量是没有限制的。()
- MUI 框架中的方法可以用在 Web 开发上。()
- MUI 框架中自带的下拉刷新和上拉加载是基于双 Webview 实现的。()
- MUI 框架中的 UI 组件的 HTML 代码必须按说明的方式设计。()
- MUI 框架中的 mui、getJSON 方法可以专门用来读取 JSON 格式数据。()

三、填空题

- MUI 项目中的页面会自动引入 .js 文件_____和 .css 文件_____。
- 检测手机是否为 iOS 系统可以使用_____。
- MUI 框架中各 UI 组件的单击使用_____事件。
- MUI 框架中实现未预加载窗口的打开采用的方法是_____。
- 自定义关闭 MUI 窗口的逻辑时,一定要重写_____方法。

四、简答题

- 试述利用 MUI 框架开发 HTML5 App 需要注意的一些事宜。
- 试述在 Chrome 中如何调试 HTML5 App。

五、编程题

使用 MUI 框架的 AJAX 通信的相关 API,完成第 7 章习题中的编程题“手机号码归属地查询”。

第 14 章

CHAPTER 14

综合实例：美食汇 App

学习目标

- 熟练掌握 manifest.json 文件的配置。
- 掌握字体图标的制作。
- 熟练掌握使用 MUI 迅速搭建 App 的界面。
- 熟练掌握 MUI 框架与服务器的通信。
- 掌握 HTML5+ 规范中的常用 API。

本章是一个整体的 App 开发,实现了一个类似“美团”的美食信息类 App。这个实例涉及 MUI 搭建界面、与服务器交互、HTML5+ API 调用等众多方面的综合知识。结合 HBuilder 的便捷,我们看到了 HTML5 技术的强大,相信它在 App 开发中一定会大放光彩。

14.1 项目介绍

本书的配套源代码中提供了一个使用 HTML5 技术开发的项目“美食汇”的源代码,同时也提供了 Android 的 APK 安装包。这个例子是使用 HBuilder 工具,基于 MUI 框架和 HTML5+ 规范的各种技术实现的。读者可以将其安装在自己的 Android 手机上,或者使用 Android 模拟器,图 14-1 是它的运行效果。

这个例子是模仿“美团”实现的一个美食信息类 App,基本上涉及了目前的 App 需要的功能,它的主要功能有向导、选项卡切换、美食列表的下拉刷新和上拉加载、地理定位、扫码、摇一摇、美食收藏和分享、订单评价、版本更新等。在本书的配套源代码中提供了该 App 开发中用到的所有图片素材和静态页面。为了让读者的学习重点集中在 App 的前端开发上,我们还特意购买了域名和租用了虚



图 14-1 美食汇的效果图

拟主机,部署了 App 开发和本书各例需要调用的服务器端 API。你可以创建好一个空白的移动 App 项目,结合本书讲解进行实践。本书以在 Android 平台上的开发为主进行讲解。

下面对各 API 的使用作一个简单的说明,实际开发中,可以结合第 7 章介绍的工具 Fiddler 来调试接口。

14.1.1 API 全局变量

为了简化和便于程序的修改,在项目的“App.js”文件中定义了一个 api_url 全局变量,用它的属性定义了 API 的地址,类似于下面的代码:

```
var api_url = {};
//通过经纬度获取所在城市 api
api_url.getCityUrl = "http://www.meishihui68.com.cn/api/geo?latitude = {0}&longitude = {1}";
//美食获取 api
api_url.getFoodUrl = "http://www.meishihui68.com.cn/api/food";
```

为了避免 url 字符串的拼接,在变量中使用了 {0}、{1}... 这样的符号表示变量的占位符,并使用原型方式为 String 字符串设计了一个 format 方法,用于变量的迅速替换,代码如下:

```
//专用于 url 字符串替换
String.prototype.format = function()
{
    if(arguments.length == 0) return this;
    for(var s = this, i = 0; i < arguments.length; i++)
        s = s.replace(new RegExp("\\{" + i + "\\}", "g"), arguments[i]);
    return s;
};
```

使用时,只需对 url 字符串变量赋值进行替换即可,代码示范如下:

```
api_url.getCityUrl.format(值 1, 值 2)
```

14.1.2 API 介绍

由于篇幅限制,服务器端的 API 就不在这里详细介绍了。在本书配套资源包中,对各 API 作了详细的说明,包括 API 的地址、调用方法、传递的数据格式和需要的报头,以及 API 调用后返回的数据。图 14-2 是这些 API 说明文档的示意图。

1. 所在城市获取API

我们利用百度地图的API制作了该API，根据移动设备获取的经度和纬度，定位所在的城市。

● 地址：

```
http://www.meishihui68.cn/api/geo?latitude={0}
&longitude={1}
```

● 请求参数：

{0}代表纬度、{1}代表经度

● 请求的方法：GET

● 返回的数据：JSON对象，格式为

```
{
  status://状态
  city://城市名
}
```

2. 美食获取API

图 14-2 API 说明文档示意图

14.2 字体图标的制作

在“美食汇”App 开发中使用了大量的字体图标。字体图标相对于普通的图片有明显的优势：

- 多个图标字体合成一个字体文件，避免每张图片都需要联网请求；
- 字体可任意缩放，而图片放大会失真；
- 字体图标可通过 CSS 改变颜色、设置阴影及透明效果。MUI 框架中已经提供了一些内置的字体图标，但是我们需要使用一些自定义的字体图标，这里使用阿里巴巴矢量字体图标库（详见 <http://www.iconfont.cn/>）来制作的。打开网站并登录后，出现图 14-3 的界面。

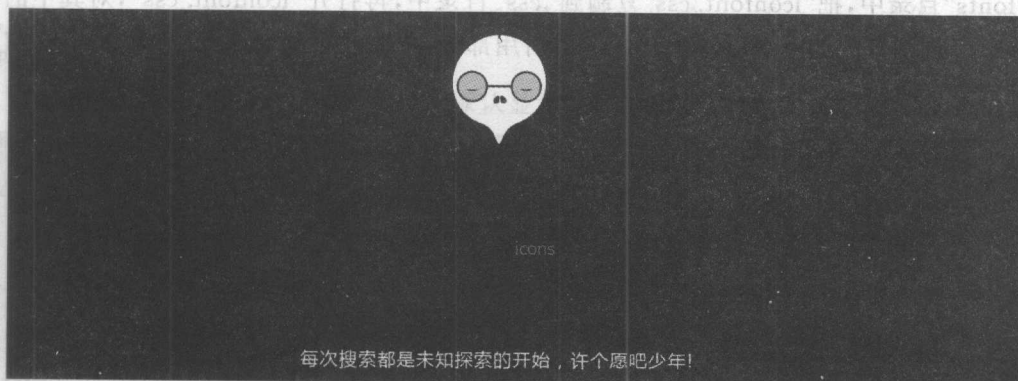


图 14-3 阿里巴巴矢量字体图标库

在页面中直接输入你需要的图标名称，例如首页，可以列出当前网站上的所有首页图标，选中你需要图标，单击“添加入库”，对其他图标可以采用类似操作。所有的图标都选择之后，单击页面右上角的“购物车”图标，网站会列表显示你所加入的所有图标，单击“添加至项目”按钮，如图 14-5 所示。



图 14-4 添加入库



图 14-5 添加入库

这里你可以创建一个新项目,例如“meishihui-icon-custom”,页面会跳转到项目管理页面,如图 14-6 所示,选择“Font class”后,可以看到预览效果,单击“下载至本地”后,会将合并后的字体文件及自动生成的 CSS 文件全部下载为一个压缩文件,解压后,文件目录如图 14-7 所示。



图 14-6 项目管理页面

我们需要复制“iconfont.ttf”文件(兼容 iOS 和 Android,只需要 ttf 字体)到移动 App 的“fonts”目录中,把“iconfont.css”复制到“css”目录中,再打开“iconfont.css”,对其中的@font-face 规则修改如下,只保留了 ttf 字体引用部分,并修改了相对路径,去除了-moz-前缀的样式。

```
@font-face {font-family: "iconfont";
  src:url('../fonts/iconfont.ttf?t=1488811549779') format('truetype');
}
.iconfont {
  font-family:"iconfont" !important;
  font-size:16px;
  font-style:normal;
  -webkit-font-smoothing: antialiased;
}
.icon-choujiang:before { content: "\e615"; }
```

使用时,需要在页面中链接“iconfont.css”,代码如下:

```
<link rel="stylesheet" href="css/iconfont.css" />
```


使用也非常简单,例如使用一个 span 标签来生成相应的图标,它的代码如下。可对它进行任意大小改变,颜色的改变,如图 14-8 所示,这比使用图片更简单。

```
<span class="iconfont icon-choujiang"></span>
```



建议一次性找到 App 中所需要的所有图标,制作好相应的字体文件和 CSS 文件。

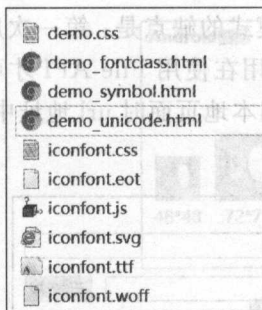


图 14-7 下载的项目



图 14-8 字体图标的不同效果

14.3 manifest.json 文件的配置

manifest.json 文件是根据 W3C 的 WebApp 规范制定的,用于对 HTML5 移动 App 打包为 ipa 或 apk 安装包进行配置,用于指定应用的显示名称、图标、应用入口文件地址及需要使用的设备权限等信息,用户可通过 HBuilder 的可视化界面视图或者源码视图来进行配置。在 HBuilder 中创建“移动 App”应用后都会在工程下生成 manifest.json 文件,在“项目管理器”中双击它即可打开。

HBuilder 打开 manifest.json 文件后默认显示“可视化视图”,可配置应用的基本信息,在它底部是各配置的选项卡,可以进行切换,如图 14-9 所示。

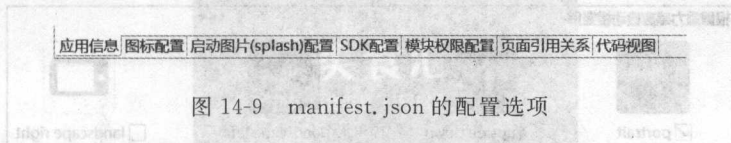


图 14-9 manifest.json 的配置选项

下面对应用信息、图标、启动图片的配置进行说明,至于 SDK 配置和模块权限配置,后文用到时再讲解。

14.3.1 应用信息配置

首先是基本信息的配置,如图 14-10 所示。

- 应用名称是指 App 打包后在手机上的快捷方式名称。
- 版本号是指应用的版本号,用户可通过 HTML5+ API 获取应用的版本号,需提交 App 云端打包后才能生效。

- 页面入口是指应用启动后自动打开的第一个 HTML 页面,可填写本地 HTML 文件地址(相对于应用根目录)或网络地址(以 http://或 https://开头)。
- 应用描述是指对 App 的一些描述,可以省略不填。
- 应用是否全屏显示: 如果勾选上,App 运行时会以全屏方式显示,手机顶部的状态栏、信号、电量等显示会自动隐藏。
- debug: 如果勾选上,不论是联真机调试,还是发布的 Apk,凡是用 console.log 输出的信息都会输出到 HBuilder IDE 的控制台。
- 应用资源是否解压: 建议选择不解压,如果选择解压,第一次启动时将解压自带资源到 SDcard,正常情况不推荐使用该模式。该模式的缺点是: 第一次启动更慢,更耗费时间,容易被第三方清理软件清理。该模式用在使用 File API 才可正常访问 www 中应用资源,以及在某些 Android rom 访问本地页面时 url 地址中包含参数的情况。

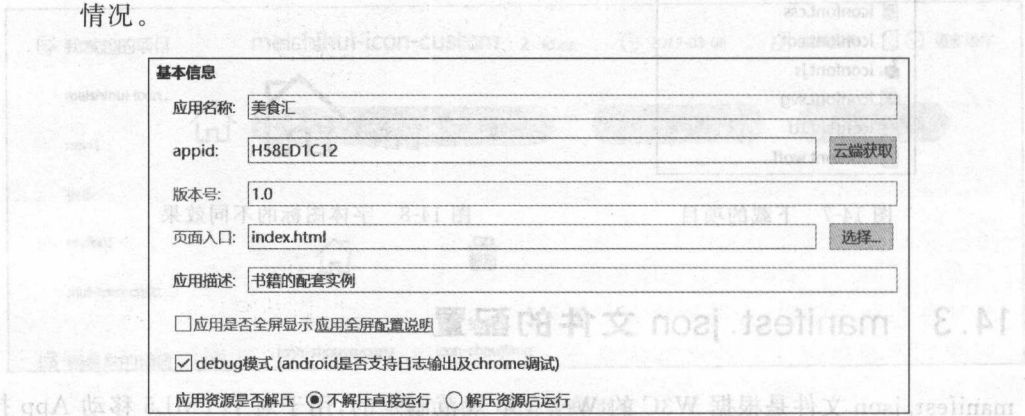


图 14-10 基本信息配置

窗口的下面是 App 重力感应配置,通过点击表示设备方向的按钮来选择设备支持重力感应旋转方向。重力选择按钮可选择一个或多个,选择多个方向后,应用可按照指定方向显示应用页面,如只选中一个按钮,表示终端只支持一个方向显示页面内容,如图 14-11 所示。

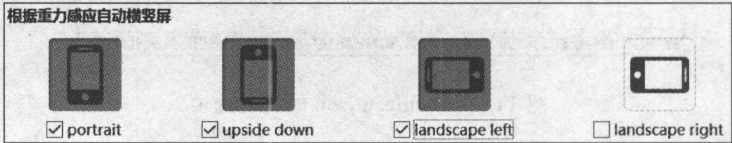


图 14-11 重力感应配置

14.3.2 图标配置

在图 14-9 中的选项卡中切换到“图标配置”,图标必须采用 png 格式图片,单击中间“+”号按钮,去选择一个大图标后,单击“自动生成所有图标并替换”按钮,IDE 会自动生成 iPhone、iPad、Android 下面的各种尺寸的图标,如图 14-13 所示,这是系统自动生成的 Android 图标。单击“浏览生成图标所在目录”按钮,可以浏览所在 Windows 目录。

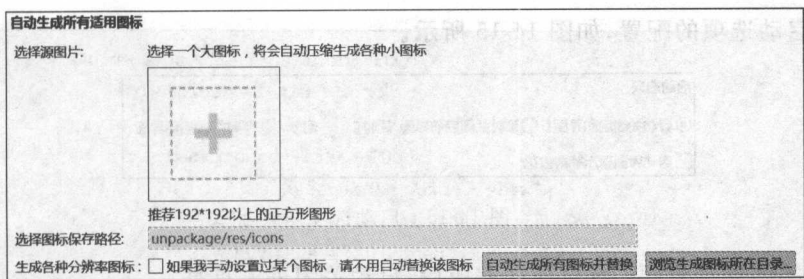


图 14-12 图标选择框

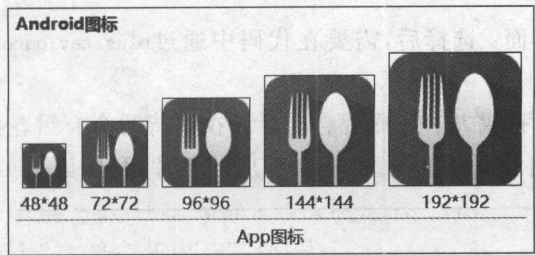


图 14-13 Android 各尺寸图标

14.3.3 启动图片配置

在图 14-9 中的选项卡中切换到“启动图片(splash)配置”,启动图片也必须采用 png 格式图片,图 14-14 是“美食汇”的启动图片,事先需要制作各种像素尺寸的图片,对于 Android 平台,需要 480×762、720×1242、1080×1882 三种像素的图片。



图 14-14 启动图片

首先是启动选项的配置,如图 14-15 所示。

启动选项

☒ 自动关闭启动界面 ☐ 延时关闭启动界面 延时: 毫秒 ☐ 手动关闭启动界面

☒ 启动界面显示等待雪花

图 14-15 启动选项

- 自动关闭启动界面: 首页加载完成后,自动关闭进入首页。
- 延时关闭启动界面: 选择后需要输入延时时间,首页加载完成后,延迟一定时间进入首页。
- 手动关闭启动界面: 选择后,需要在代码中通过 `plus.navigator.closeSplashScreen()` 方法手动关闭,否则将一直显示。
- 启动界面显示等待雪花: 选择后启动界面在等待时会有雪花(旋转进度)显示。

在页面底部,根据启动图片的类型,选择相应的图片即可,如图 14-16 所示。

Android启动图片设置

注: Android屏幕类型很多,如果未手动设置其他分辨率,遇到未不包括时,Android系统会自动选取临近分辨率图片显示。如果应用非全屏,启动图片高度为屏幕高度减掉状态栏高度。




启动图片类型	操作
高分屏启动图片 480x800	 <input type="button" value="选择"/> <input type="button" value="还原默认"/>
720P高分屏启动图片 720x1280	 <input type="button" value="选择"/> <input type="button" value="还原默认"/>
1080P高分屏启动图片 1080x1920	 <input type="button" value="选择"/> <input type="button" value="还原默认"/>

图 14-16 Android 启动图片设置



manifest.json 文件修改保存后,调试中必须重启程序。App 的版本号、图标和启动图片的设置必须是 App 打包后才能看到效果。

14.4 向导

向导是 App 常见的功能,当一个 App 安装后第一次启动时,用户会看到向导窗口,通过向导页的左右滑动,它可以很清晰地展示系统的一些功能特性,或者对 App 作一些介绍。图 14-17 是“美食汇”中的向导(guide.html)运行效果,一共是 4 个滑动项,当用户滑动到最后一项,单击“立即体验”按钮后,进入 App 的首页。为了增强效果,使用了 CSS 对文字的进入设计了一些特效。

4 个滑动项的设计相对比较简单,使用的是 MUI 框架中的轮播组件。为了更好地适配手机,4 张图片是使用背景图片方式进行的展现,下面是 HTML 代码的设计:

```
<div id="slider" class="mui-slider mui-fullscreen">
  <div class="mui-slider-group">
    <!-- 第一张 -->
```

```

<div class="mui-slider-item">
  <div class="item-logo"
    style="background: url(imgs/guide01.png) no-repeat;
    background-size: 100% 100%;">
    <div class="animate-guide-show">
      <h2 class="animated bounceInDown">网罗美食</h2>
      <li class="animated bounceInLeft">
        尽享天下美食</li>
      <li class="animated bounceInRight">
        体会百味生活</li>
    </div>
  </div>
</div>
<!-- 第二张,第三张,第四张和第一张设计类似,只是图片不同 -->
<!-- 滑动指引圆点 -->
<div class="mui-slider-indicator">
  <div class="mui-indicator mui-active"></div>
  <div class="mui-indicator"></div>
  <div class="mui-indicator"></div>
  <div class="mui-indicator"></div>
</div>
</div>

```

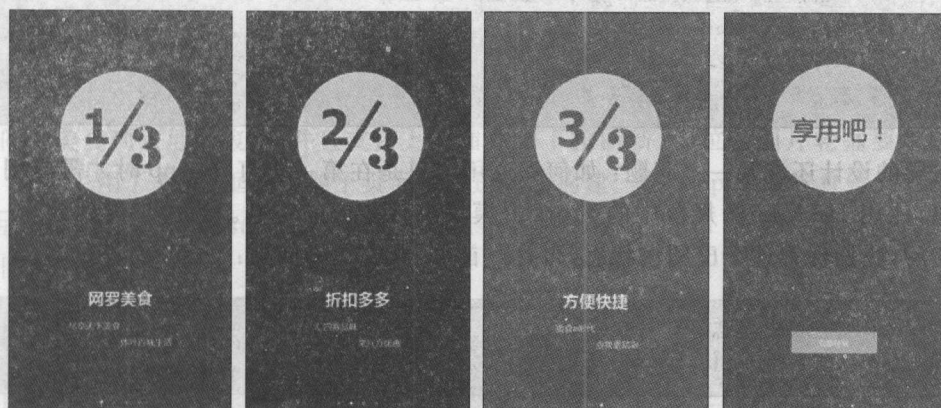


图 14-17 向导页效果图

对于文字进入的特效,使用了 CSS 中的动画。我们以文字从上面进入的特效为例,它的动画规则设计如下:

```

@-webkit-keyframes bounceInDown {
  0%, 60%, 75%, 90%, 100% {
    -webkit-animation-timing-function:
      cubic-bezier(0.215, 0.610, 0.355, 1.000);
  }
  0% {

```

```

        opacity: 0;
        -webkit-transform: translate3d(0, -3000px, 0);
    }
    60% {
        opacity: 1;
        -webkit-transform: translate3d(0, 25px, 0);
    }
    75% {
        -webkit-transform: translate3d(0, -5px, 0);
    }
    90% {
        -webkit-transform: translate3d(0, 3px, 0);
    }
    100% {
        -webkit-transform: none;
    }
}

```

为了能应用这个动画规则,又特意设计了相应的样式选择器,便于使用 JavaScript 的 DOM 操作动态地添加或移除动画,代码如下:

```

.guide-show.bounceInDown {
    -webkit-animation-name: bounceInDown;
    -webkit-animation-play-state: running;
    -webkit-animation-delay: 1s;
    display: block;
}

```

向导的设计还涉及一个问题:如何保证用户只是在第一次打开 App 时才需要浏览它?在这里,使用 HTML5+规范中的 Storage 模块实现相应的需求。当用户滑到最后一项点击按钮后,我们保存这个已阅读状态,再打开首页,核心代码如下:

```

//写入向导已浏览标志
plus.storage.setItem("launchFlag", "true");

```

在 App 的入口点(index.html)页面中对这个状态作了判断,如果为 true,说明已经浏览过,直接打开首页,否则是第一次启动 App,打开“向导”窗口,核心代码如下:

```

//读取向导页阅读状态
var launchFlag = plus.storage.getItem("launchFlag");
if(launchFlag){
    //打开首页
    mui.openWindow({
        url:"main.html",
        id:"main.html"
    });
}

```



```
else{
    //打开向导页
    mui.openWindow({
        url:"guide.html",
        id:"guide.html"
    })
}
```

14.5 首页

首页是“美食汇”App 最核心的窗口,为了提高用户体验,在入口页面(也就是 index.html)中使用了预加载技术,预加载的使用如下:

```
mui.init({
    preloadPages:[{
        url:"main.html",
        id:"main.html"
    }]
});
```

14.5.1 使用子页面构建首页

首页是整个 App 设计中最为复杂和关键的窗口,头部搜索框和底部的选项卡是固定的,中间的美食列表可以滚动,并且支持上拉加载和下拉刷新,上拉时图片轮播要求能自动隐藏。首页采用了 MUI 框架中的子页面技术进行了构建。在 App 中最终看到的首页实际上是由 main.html、foodlist_main.html、foodlist_pullrefresh.html 三张页面组合构建而成的,如图 14-18 所示。



图 14-18 子页面构建首页

在 main.html 中,先使用子页面技术,将 foodlist_main.html 作为子页面附加,核心代码如下:

```
var subpages = ['foodlist_main.html', 'settings.html'];
var subpage_style = {
    top: '0',
    bottom: '51px'
};
...
var sub = plus.webview.create(subpages[i], subpages[i], subpage_style);
...
self.Append(sub);
```

在 foodlist_main.html 中,又将 foodlist_pullrefresh.html 作为子页面附加:

```
mui.init({
    //子页配置
    subpages: [{
        url: 'foodlist_pullrefresh.html',
        id: 'foodlist_pullrefresh.html',
        styles: {
            top: '175px',
            bottom: '0px',
        }
    }]
});
```

14.5.2 美食列表数据的请求和刷新

“美食汇”中的所有数据交互都采用 MUI 框架中的 AJAX 技术实现的。首页中的数据请求是整个 App 中最典型的,我们以它为例,介绍“美食汇”App 中的数据交互。foodlist_pullrefresh.html 中有如下核心代码,addFood 方法用于实现上拉或下拉刷新的通用。

```
function addFood(isUp){
    mui.ajax(api_url.getFoodUrl, {
        type: "POST",
        timeout: 10000,
        data: JSON.stringify(meishiApp.dataType.foodData),
        headers: { 'Content-Type': 'Application/json' },
        success: function(res) {
            //数据读取成功,根据上拉或下拉生成列表,代码省略...
        },
        error: function(xhr, type, errorThrown) {
            //异常处理;
            console.log(type);
        }
    })
}
```

```
});
}
```

在这个数据请求中展示了如何使用 MUI 的 AJAX 技术进行数据通信的正确方式,即请求的方法、请求的数据、请求需要的额外报头,当然请求的 API 地址也必须构建正确。这个需要仔细阅读 API 说明。

对于上拉和下拉刷新,根据 MUI 的说明文档,在 foodlist_pullrefresh.html 中进行了相应的配置,如下:

```
//配置下拉和上滑刷新
mui('#scroll').pullToRefresh({
  down: {
    callback: pulldownRefresh //相应的回调函数名
  },
  up: {
    auto:true,
    contentrefresh: '美食赶来中...',
    contentnomore: '没有更多美食了',
    callback: pullupRefresh //相应的回调函数名
  }
});
```

目前,为了降低开发难度,HBuilder 内置了一些常用的模板,可以在项目中新建文件,选择“HTML 文件”后,使用 HBuilder 内置的“mui 下拉刷新、上拉加载”模板直接制作。

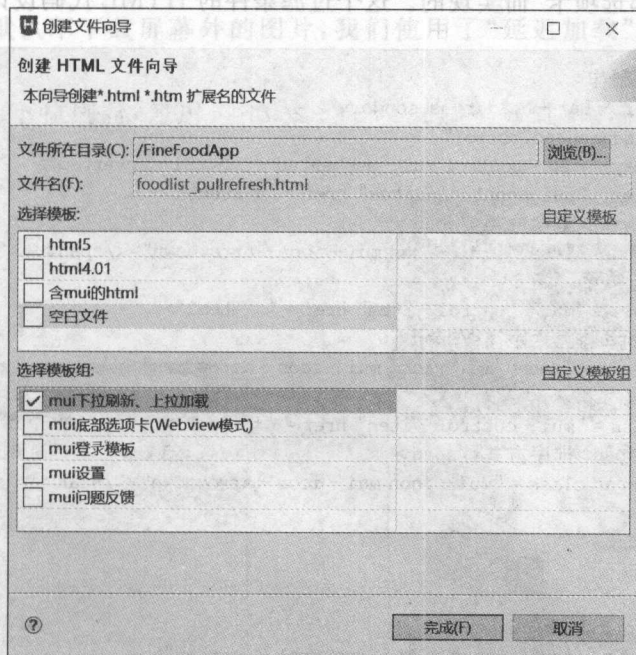


图 14-19 下拉刷新、上拉加载模板



建议使用 HBuilder 中内置模板建立文件时, 最好在另外一个空项目中制作, 设计完成后再把相关文件复制回工程项目。

至于美食生成的图文列表, 是以 MUI 框架中的图文列表为基础的, 在 14.9.1 节中会讲解列表生成的技术和思路。

14.5.3 滑动手势的处理

在向上滑动过程中, 希望图片轮播会自动隐藏, 隐藏需要使用 DOM 操作, 这个比较容易实现, 关键是手势的处理, 我们在 foodlist_pullrefresh.html 中处理如下。

```
mui.init({
    gestureConfig: {
        swipe: true //启用滑动手势
    }
});

//向上滑动手势
window.addEventListener("swipeup", function() {
    //隐藏轮播, 代码略 ...
});
```

14.5.4 过滤条件的制作

在首页的效果中设计了一个漂亮的过滤条件效果, 如图 14-20 所示, 这是借用了 MUI 演示项目中的“顶部选项卡”而实现的。这个过滤条件的 HTML 代码设计如下:

```
//过滤条件头部制作
<div class="mui-bar-header-secondary">
    <div id="segmentedControl"
class="mui-segmented-control mui-segmented-control-inverted">
        <a class="mui-control-item" href="#item1">
            <span>美食分类</span>
            <span class="mui-icon mui-icon-arrowdown"></span>
        </a>
        <a class="mui-control-item" href="#item2">
            <span>美食距离</span>
            <span class="mui-icon mui-icon-arrowdown"></span>
        </a>
        <a class="mui-control-item" href="#item3">
            <span>排序方式</span>
            <span class="mui-icon mui-icon-arrowdown"></span>
        </a>
    </div>
</div>

//过滤明细制作
<div class="dp_filter_content">
    <div id="item1" class="mui-control-content">
        <div id="filter_scroll" class="mui-scroll-wrapper">
```

```
<div class="mui-scroll">
  <ul class="mui-table-view" tabid="0">
    <li class="mui-table-view-cell">
      /* 分类过滤明细,略 */
    </li>
  </ul>
</div>
</div>
<div id="item2" class="mui-control-content">
  <div id="filter_scroll" class="mui-scroll-wrApper">
    <div class="mui-scroll">
      <ul class="mui-table-view" tabid="1">
        <li class="mui-table-view-cell">
          /* 距离过滤明细,略 */
        </li>
      </ul>
    </div>
  </div>
</div>
<div id="item3" class="mui-control-content">
  <div id="filter_scroll" class="mui-scroll-wrApper">
    <div class="mui-scroll">
      <ul class="mui-table-view" tabid="2">
        <li class="mui-table-view-cell">
          /* 排序方式过滤明细,略 */
        </li>
      </ul>
    </div>
  </div>
</div>
```

14.5.5 MUI 插件的使用

在首页的列表滑动效果中,图片较大或网络通信较差时,为了能加快页面渲染速度、提升页面滚动性能、默认不下载屏幕外的图片,我们使用了“延迟加载”插件,使用方式见 13.10.1 节。

对于首页中的下拉刷新,使用了 13.10.4 节中的“单页面刷新插件”,运行效果如图 14-21 所示。

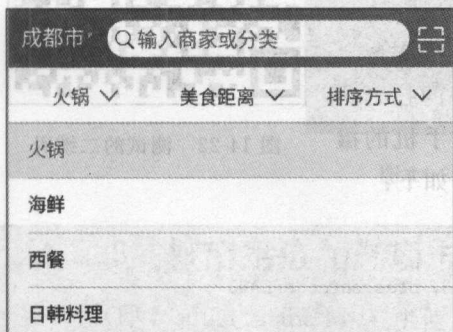


图 14-20 过滤条件



图 14-21 下拉刷新效果

14.5.6 窗口数据的传递

数据在页面之间的传递在 App 开发中也是常见的场景。在首页中单击每条美食列表, App 会自动打开美食详情页面(即 food_detail.html),将显示该美食的一些详细信息。这部分的核心代码如下:

```
//美食列表选项的单击
mui("#foodlist").on("tap", "li", function() {
    var id = this.getAttribute("data-id");
    //打开美食详情
    mui.openWindow({
        id: 'food_detail.html',
        url: 'food_detail.html',
        show: {
            autoShow: false    //控制详情窗口不要自动打开
        },
        extras: {deal_id: id}    //extras 传递的数据
    });
});
```

在这里,美食的编号是事先用 HTML5 中的自定义属性“data-id”写入列表项的,为了改善用户体验,同时控制了详情窗口不能自动显示,显示前自动显示系统等待对话框。



如果数据需要连续在多个页面间传递,建议不要采用对象形式,可以把数据对象序列化成字符串,再作为对象的属性值进行传递。

14.5.7 扫码的实现

扫码功能是目前众多 App 中常见的一个功能,用户使用手机的摄像头就可以迅速关注微信、打开页面甚至收付款等。在首页也实现了一个扫码功能,图 14-22 中提供了本书的一个配套 URL 地址(即 http://www.meishihui68.com.cn/staticpage/food_detail.html)的二维码,App 识别到这个二维码后,会自动打开一个固定的美食详情页面。

在首页单击扫码按钮后,会打开扫码窗口(barcode_scan.html),自动出现扫码控件,用手机对准该二维码,可以实现自动识别,如果是其他二维码,只会弹出结果。窗口中的扫码功能是使用 HTML5+规范中的 BarCode 模块,调用手机的摄像头对条码图片进行扫描处理的。其中的核心代码如下:



图 14-22 测试的二维码

```
//识别格式设定
var filter = [plus.barcode.QR, plus.barcode.EAN13, plus.barcode.EAN8];
//扫码控件样式设定
```



```

var style = { frameColor: '#00FF00', scanbarColor: '#00FF00' };
//生成控件,bcid是页面上容器的id号
scan = new plus.barcode.Barcode('bcid', filter, style);
//识别成功调用函数
scan.onmarked = onmarked; //onmarked为识别成功回调函数名
//开始识别,成功后保存二维码截图在 barcode 目录中
scan.start({ conserve: true, filename: '_doc/barcode/' });

```

扫码页面的效果如图 14-23 所示,除了扫描外部的二维码,还可以从手机相册选择相应的二维码图片进行识别,从相册选择图片的核心代码如下,主要使用了 HTML5+ 规范中的 Gallery 模块。

```

// 从相册中选择二维码图片
plus.gallery.pick(function(path) {
    plus.barcode.scan(path, onmarked,
        function(error) {
            plus.nativeUI.alert('无法识别此图片'
                );
        });
}, function(err) {
    plus.nativeUI.
        alert('Failed: ' + err.message);
});

```



图 14-23 扫码窗口效果

14.5.8 城市定位和选项卡切换

在首页(即 foodlist_main.html)中使用了 HTML5+ 的 Geolocation 模块获取手机的纬度和经度,再基于我们提供的 API(Baidu 地图的 API 作的封装),传入纬度和经度后实现了

城市定位功能。虽然 W3C 制定的 HTML5 标准中已经提供了标准的地理信息定位功能,但在某些平台存在差异或未实现,HTML5+ 规范的这个模块则保持了各平台的统一性。下面是获取纬度和经度的核心代码:

```
if(window.plus){
    plus.geolocation.getCurrentPosition(function(p) {
        //p.coords.latitude 是纬度,p.coords.longitude 是经度
        var url = api_url.getCityUrl.format(p.coords.latitude,
            p.coords.longitude);
        //下面是 API 调用,代码略
    }
    ...
}
```

选项卡切换的思路与例 12-5 中的思路是类似的,不同的是“抽奖”和“我的”这两个选项卡对应的页面不是使用子页面附加在 main.html 页面上的,而是直接创建并展示出来的。为了简化开发,同样可以新建,选择“HTML”文件,选择 HBuilder 内置的模板“mui 底部选项卡(Webview 模式)”进行页面快速生成后,在此基础上修改,如图 14-24 所示。

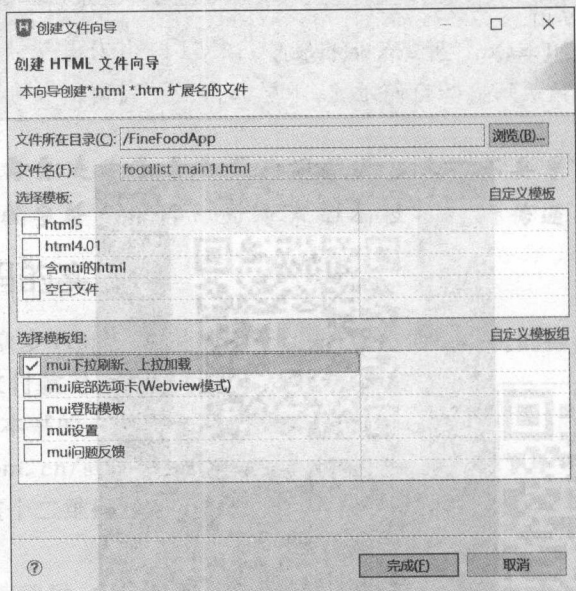


图 14-24 底部选项卡的创建

14.6 美食详情

美食详情窗口(即 food_detail.html)从首页接收到传递过来的美食 ID 后,与提供的 API 实现交互,得到美食的各详细信息,再利用 DOM 操作将数据显示在页面上,这个操作比较简单,不再赘述。页面上取得从首页传递的美食 ID 的核心代码如下,更多细节可参考 13.7 节中关于“窗口之间数据的传递”。

```
mui.plusReady(function() {  
    var self = plus.webview.currentWebview();  
    //deal_id是首页传递过来的参数  
    var url = api_url.getFoodDetailByIdUrl.format(self.deal_id);  
    //下面是和服务器交互的代码,略...  
}
```

下面简要讲述美食详情窗口上的拨打电话、百度地图定位显示、评论中的图片预览、分享、收藏功能的实现思路。

14.6.1 拨打电话

在美食详情窗口中,单击信息中的电话号码,App会自动打开手机的拨号功能,并将电话号码自动输入,用户单击手机拨打按钮后再开始拨打电话。这个功能的实现是基于HTML5+规范的Device模块实现的,代码如下:

```
plus.device.dial("电话号码", true);  
//如果是 false,则会直接拨打电话
```

14.6.2 百度地图定位显示

在美食详情窗口中,单击美食信息中的地址后,App会自动打开窗口(map.html),在该窗口中使用Baidu地图显示美食所在的地址。效果如图14-25所示。如果单击“百度地图”,则会打开手机已安装的“百度地图”,实现到该位置的导航,如图14-26所示。



图 14-25 百度地图显示效果

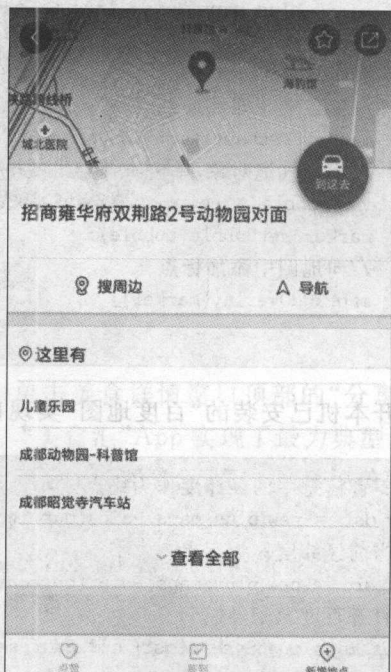


图 14-26 百度地图

要实现这个效果,我们首先必须申请百度地图的密钥,具体步骤可以参考官方的网址(<http://ask.dcloud.net.cn/article/29>)。申请之后,打开“manifest.json”文件,切换图 14-9 中的“SDK 配置”,对地图进行配置,分别输入申请到的百度地图的 iOS 和 Android 的密钥,如图 14-27 所示。

图 14-27 百度地图密钥配置

这里主要是应用了 HTML5+ 的 Map 模块,核心的代码如下:

```
var pcenter = new plus.maps.Point(cur.lng, cur.lat);
setTimeout(function() {
    //生成 map 对象,"map"是页面上充当容器的对象 ID 值
    map = new plus.maps.Map("map");
    //设置地图初始中心点和缩放级别
    map.centerAndZoom(pcenter, 17);
    createMarker();
}, 300);

function createMarker() {
    //生成地图上的标点对象,经度和纬度由详情页面传入
    var marker = new
        plus.maps.Marker(new plus.maps.Point(cur.lng, cur.lat));
    //标点图标
    marker.setIcon("imgs/map.jpg");
    //标点文字即地址
    marker.setLabel(cur.address);
    //生成气泡对象,单击图标会出现电话
    var bubble = new plus.maps.Bubble(cur.phone);
    marker.setBubble(bubble);
    //向地图中添加标点
    map.addOverlay(marker);
}
```

打开本机已安装的“百度地图”实现自动导航,核心的代码如下:

```
//美食信息中的经纬度作为终点
var dst = new plus.maps.Point(cur.lng, cur.lat);
//当前经纬度作为起点
var src = new plus.maps.Point(currentlng, currentlng);
//打开百度地图导航
plus.maps.openSysMap(dst, cur.address, src);
```

14.6.3 评论中的图片预览

在美食详情窗口的底部,每条美食详情都会得到服务器生成的两条评论,单击“食友点评”会进入评论列表窗口,单击评论中的图片之后都会进入图片预览界面,这是使用 13.10 节中介绍的 MUI 框架的图片预览插件实现的。效果如图 14-28 所示,预览进入全屏显示,并可以左右滑动,按后退键后返回美食详情窗口。

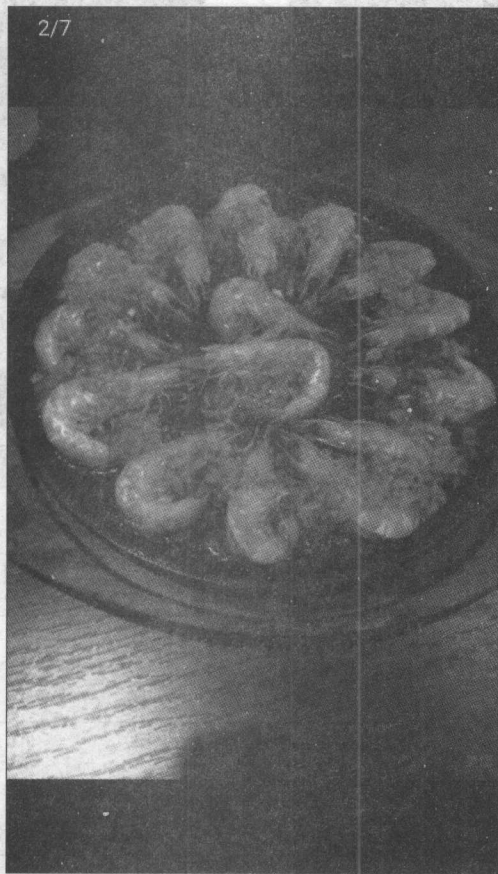


图 14-28 图片预览

14.6.4 分享

目前,分享基本上成了所有 App 的标配功能,当单击美食详情窗口顶部的“分享”按钮后,窗口底部会自动弹出分享界面,如图 14-29 所示。“美食汇”App 实现了最为典型的微信分享和系统分享功能。微信分享必须向腾讯公司申请密钥和配置 SDK,而系统分享不需要,但这种方式不支持微信的朋友圈分享,效果如图 14-30 所示。

微信的密钥申请可以参考官方的文档(<http://ask.dcloud.net.cn/article/208>),申请成功后需在 manifest.json 文件中打开选项卡“SDK 配置”配置,如图 14-31 所示。

底部弹出的分享选择效果是使用 MUI 框架中 HTML 定制的 actionsheet 实现的,核心代码如下:



图 14-29 分享选择



图 14-30 系统分享

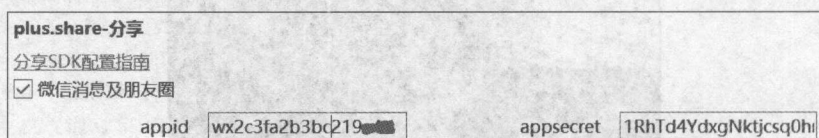


图 14-31 分享配置

```
<div id="sharediv" class="mui-popover
    mui-popover-action mui-popover-bottom">
  <ul class="mui-table-view">
    <div class="share_zone" data-class="weixin">
      
      <div class="share_des">微信好友</div>
    </div>
    <div class="share_zone" data-class="friend">
      
      <div class="share_des">朋友圈</div>
    </div>
  </ul>
</div>
```



```

        //其他选项基本类似,代码略...
    </ul>
</div>

```

要实现分享功能,需要借助 HTML5+规范中的 Share 模块,首先需要创建社交分享管理对象 ShareService,代码如下:

```

var shares = null;
updateSerivces();
//更新分享服务
function updateSerivces() {
    plus.share.getServices(function(s) {
        shares = {};
        for(var i in s) {
            var t = s[i];
            shares[t.id] = t;
        }
    }, function(e) {
        console.log("获取分享服务列表失败:" + e.message);
    });
}

```

ShareService 对象会在 manifest.json 中对于分享的配置进行读取,创建相应的属性,下面是这个对象的内容,可以看到当配置好微信分享 SDK,它的 authenticated 属性自动设置为 true:

```

{
  "sinaweibo": {
    "id": "sinaweibo",
    "description": "新浪微博",
    "authenticated": false,
    "accessToken": "", "nativeClient": false,
    "tencentweibo": {
      "id": "tencentweibo",
      "description": "腾讯微博",
      "authenticated": false,
      "accessToken": "", "nativeClient": false,
      "qq": {
        "id": "qq",
        "description": "QQ",
        "authenticated": false,
        "accessToken": "", "nativeClient": true,
        "weixin": {
          "id": "weixin",
          "description": "微信",
          "authenticated": true,
          "accessToken": "", "nativeClient": true
        }
      }
    }
  }
}

```

微信分享的数据格式构造如下：

```
{
  "content": 分享的文字内容,
  "extra": {"scene": "WXSceneSession"}, //朋友圈则为 WXSceneTimeline
  "href": 单击后的超链接,
  "title": 标题,
  "thumbs": 分享图片 url 字符串数组,
  "pictures": ["_www/logo.png"] //这里是 App 的 Logo 地址
}
```

基于 ShareService 对象和微信分享的数据格式,可以完成微信分享的代码如下：

```
//微信分享
function shareAction(sharetype) {
  //构建微信要分享的消息格式
  var msg = {};
  msg.content = food_detail.food_desc.innerText;
  msg.extra = {};
  if(sharetype == "weixin") {
    msg.extra.scene = "WXSceneSession";
  } else if(sharetype == "friend") {
    msg.extra.scene = "WXSceneTimeline";
  } else {
    return;
  }
  msg.href = "http://www.meishihui68.com.cn
              /staticpage/food_detail.html";
  msg.title = food_detail.foodtitle.innerText;
  var thumbs = [];
  thumbs.push(food_detail.foodimg.src);
  msg.thumbs = thumbs;
  msg.pictures = ["_www/logo.png"];
  //取微信的授权
  sharetype = "weixin";
  var share = shares[sharetype];
  //已授权
  if(share.authenticated) {
    share.send(msg, function() {
      mui.toast("分享成功!");
      mui('#sharediv').popover('toggle');
    }, function() {
      mui.toast("分享失败");
    });
  }
}
```

系统分享是基于 HTML5+规范中的 Native.js,调用了 Andorid 的原生代码实现的,调用 Andorid 代码的方式如下：


```

//调用 Android 的系统分享
function share(shareTip, shareText) {
    if(plus.os.name == "Android") {
        //导入 Java 类对象
        var Context = plus.android.importClass("android.content.Intent");
        //获取应用主 Activity
        var Main = plus.android.runtimeMainActivity();
        //将类 Context 的这个行为(Action)ACTION_SEND,赋给 shareIntent
        var shareIntent = new Context(Context.ACTION_SEND);
        //设置分享类型
        shareIntent.setType("text/plain");
        //设置分享文本
        shareIntent.putExtra(Context.EXTRA_TEXT, shareText);
        Main.startActivity(Context.createChooser(shareIntent, shareTip));
    }
}

```

从这里可以看出,Native.js 功能非常强大,它可以无障碍地调用系统的 API,从另外一个角度讲,HTML5+规范的可扩展能力在 Native.js 的基础上得到了大大提高。这里只是使用 Android 平台下的系统分享简单的文本和超链接。

14.6.5 收藏

下面介绍美食收藏功能实现的思路,它是使用 HTML5+规范中的 Storage 模块管理应用本地数据存储的,其实这里也可以使用 HTML5 标准的 localStorage 实现。进入窗口时,要先读取原有的数据存储,如果不存在,则要创建空对象,代码如下:

```

food_detail.favourdata =
    JSON.parse(plus.storage.getItem("favourdata")) || [];

```

添加、移除以及进入窗口时,都要注意切换窗口顶部“收藏”按钮的图标状态。

```

//添加到收藏列表
function addToFavour() {
    food_detail.favourdata.push(food_detail.thedata);
    plus.storage.setItem("favourdata",
        JSON.stringify(food_detail.favourdata));
    //切换图标
    food_detail.collect.classList.remove("icon-shoucang1");
    food_detail.collect.classList.add("icon-shoucang");
    plus.nativeUI.toast("收藏成功!");
}
//从收藏列表移除
function removeFromFavour() {
    meishiApp.removeFavour(self.deal_id);
    food_detail.collect.classList.remove("icon-shoucang");
}

```



```
food_detail.collect.classList.add("icon-shoucang1");  
plus.nativeUI.toast("收藏已取消!");  
}
```

用户收藏美食信息后,可以在首页窗口的底部选项卡中选择“我的”,打开 App 的侧滑菜单(mine-offcanvasmenu.html,实现的原理见例 12-4),效果如图 14-32 所示,单击“我的收藏”后进入“我的收藏”窗口(myfavour_list.html)。该窗口中从本地存储中读取所有的收藏信息生成了列表,并在头部显示出收藏的美食条数。当用户在某个列表上手指向左滑动时,列表项上会出现相应的“删除”按钮,单击后会自动移除这条美食信息。移除包含了两层含义,一是从页面上使用 DOM 操作删除该节点对象,二是要将本地存储中的相应对象删除。滑动出现删除项的效果可以参考 13.8.4 节的内容,窗口执行的效果如图 14-33 所示。



图 14-32 侧滑菜单



图 14-33 移除列表

14.7 抽奖

“美食汇”App 中提供了一个常见的抽奖功能(winlottery.html),用户在手机上打开窗口后,点击屏幕或使用“摇一摇”的功能,可以获得一个随机的金额奖励,每天只能抽奖一次,如果未抽中,界面也会有相应提示,提示使用了自定义窗口(lottery_win.html)。单击顶部的“中奖记录”后,会打开用户的“中奖记录”窗口(prize_list.html)。效果如图 14-34 所示。

14.7.1 授权打开窗口

在 App 中会经常碰到这样的场景,有些窗口只能提供给已登录的用户使用,“美食汇”中的抽奖、我的订单这两个窗口就必须先登录。使用时,如果未登录过,App 自动打开登录窗口,登录成功以后,App 自动打开相应的窗口(相当于以前在 Web 开发中常用的页面授权和重定向)。为了实现这个效果,我们在 App.js 中实现了自定义的全局方法,要打开的窗口 URL 也会作为参数传递给登录窗口。



图 14-34 抽奖效果

```
/* 进行授权打开页面,没有 token 的一律先转向登录,data 是需要传递的数据对象 */
```

```
meishiApp.openAuthorWindow = function(win, data) {
```

```
    if(!data) {data = {};};
```

```
    data.win = win;
```

```
    //这里是取出用户的 token
```

```
    var token = this.GetUserToken();
```

```
    //没有 token
```

```
    if(!token) {
```

```
        mui.openWindow({
```

```
            id: 'login.html',
```

```
            url: 'login.html',
```

```
            extras:{data:JSON.stringify(data)}
```

```
        });
```

```
    }
```

```
    else{
```

```
        mui.openWindow({
```

```
            id: win,
```

```
            url: win,
```

```
            extras:{data:JSON.stringify(data)}
```

```
        });
```

```
    }
```

```
}
```

14.7.2 界面处理

在抽奖窗口中,有两个页面的细节需要处理——背景要能实现自动适配;要实现摇手的动画。

背景图片是事先用高分辨率制作的,利用了 CSS 的背景图片相关属性:

```
background: url(imgs/swing_red_packet_bg.png) no-repeat;
background-size: 100% 100%;
```

摇一摇的动画效果是借助于 CSS 的动画规则实现的,当单击或摇晃手机时,使用 DOM 操作动态地为图片添加样式。

```
.shake_hand {animation: shakehand 0.6s 2 ease;}
/* 动画规则 */
@keyframes shakehand {
  0% {-webkit-transform: rotateZ(0);}
  50% {-webkit-transform: rotateZ(-35deg);}
  100% {-webkit-transform: rotateZ(35deg);}
}
```

14.7.3 摇一摇

摇一摇功能实现是使用了 HTML5+ 规范中的 Accelerometer 模块和 Audio 模块。前者是用于监听设备加速度传感器,后者是用于播放摇动的音频。

```
var p = null; //播放的音频对象
var wid = null; //监听 id
function prepareHandShake() {
  // 监听加速度
  wid = plus.accelerometer.watchAcceleration(function(a) {
    if(!p && (Math.abs(a.xAxis) +
      Math.abs(a.yAxis) + Math.abs(a.zAxis) > 30)) {
      shakeLottery();
    }
  }, function(e) {
    console.log("watch failed:" + e.message)
  }, {
    frequency: 100
  });
}
function shakeLottery() {
  //播放音频
  p = plus.audio.createPlayer("_www/audio/shake.wav");
  p.play();
  //播放动画
  hand.classList.add("shake_hand");
  //振动 2 秒钟
  plus.device.vibrate(400);
  //停止播放
  setTimeout(function() {
```



```

        if(p) p.stop();
        delete p;
        p = null;
    }, 2000);
}

```

抽奖结果是在摇一摇的动画结束时出现的,这里利用了 HTML5 的“animationend”(动画结束)事件:

```

//播放完摇手动画后读取数据
win.hand.addEventListener("animationend", function() {
    //移除动画样式
    hand.classList.remove("shake_hand");
    floatWebview();    //打开抽奖结果窗口
});

```

14.7.4 自定义窗口

自定义窗口是 App 开发中的常用需求,抽奖结果(lottery_win.html)是自定义窗口,主要是利用了 WebView 的一些特性,它和普通窗口的制作没有什么不同,只是显示时,控制它的窗口大小和其他一些属性而已。下面是创建自定义窗口的代码:

```

function floatWebview() {
    // 避免快速多次单击创建多个窗口
    if(floatw) {return;}
    floatw = plus.webview.create("lottery_win.html",
        "lottery_win.html", {
        width: '240px',
        height: '356px',
        margin: "auto",
        background: "rgba(0,0,0,0.4)", //控制背景
        scrollIndicator: 'none',        //不显示滚动条
        scalable: false,                //不能缩放大小
        popGesture: 'none'              //关闭手势
    });
    floatw.addEventListener("loaded", function() {
        floatw = null;
    }, false);
}

```

14.7.5 跨页面调用方法

抽奖结果窗口显示以后,在关闭窗口以前,还涉及将抽奖窗口上的相关信息进行更新的问题,为此,我们利用 MUI 框架的另外一种方法,在抽奖窗口(winlottery.html)上设计了一个方法 updateLotteryInfo,代码如下:

```
//添加自定义方法,供其他 Webview 调用
document.addEventListener("updateLotteryInfo", function(event) {
    //取抽奖金额数据
    var amount = event.detail.amount;
    //更新页面的相关信息,代码略...
})
```

在抽奖结果窗口 (lottery_win.html) 中,使用了下列方法直接调用在上面代码中的 updateLotteryInfo 方法:

```
var op = null;
op = ws.opener(); //取得抽奖窗口 WebView 对象
mui.fire(op, "updateLotteryInfo", {amount:prizeAmount}); //调用并传递金额
```

14.8 注册和登录

“美食汇”App 也提供了注册 (login.html) 和登录 (reg.html) 功能,这两个功能的完成和普通 Web 开发没有什么区别,不同的是没有 form 表单,数据交互是依赖于 MUI 框架中的 AJAX 通信。

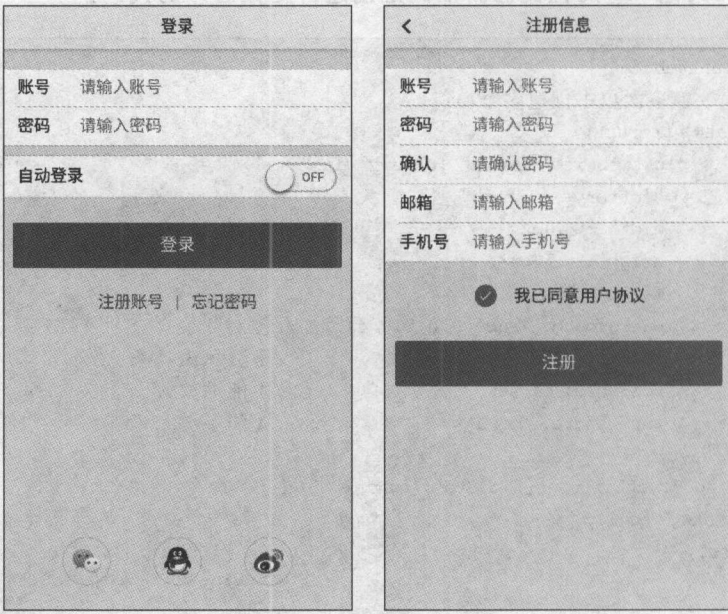


图 14-35 登录和注册效果

每次登录成功后,服务器端都会返回一个用户 token(随机的 GUID 字符串)用于标识用户,接收到数据后,token 使用 Storage 模块进行保存,同时自动更新侧滑菜单中用户的头像等信息。对于自动登录问题,我们也是使用 Storage 存储了它的状态,并在首页 (main.html) 中重新定义了系统的 back 方法,在这个方法中,根据状态位决定是否移除用户 token,

代码如下：

```
var oldback = mui.back;
mui.back = function(){
    mui.confirm("你确定要退出?", "温馨提示",
        ["确定", "点错了"], function(e){
            if(e.index == 0){
                //取出配置
                var settings = meishiApp.getSettings();
                //如果不是自动登录, 移除 token 数据
                if(!settings.autoLogin){
                    meishiApp.removeUserToken();
                }
                oldback();
            }
        });
}
```

登录和注册中输入的数据都需要借助于正则表达式。在“App.js”中把需要使用的正则表达式都定义为全局变量使用,以注册为例,它的验证是这样的:

```
function valideUser() {
    if(!meishiApp.regExp.notEmptyReg.test(reg.account.value)) {
        plus.nativeUI.toast('请输入账号');
        return false;
    }
    if(!meishiApp.regExp.notEmptyReg.test(reg.pass.value)) {
        plus.nativeUI.toast('请输入密码');
        return false;
    }
    if(!meishiApp.regExp
        .notEmptyReg.test(reg.confirmpass.value)) {
        plus.nativeUI.toast('请确认密码');
        return false;
    }
    if(reg.confirmpass.value != reg.pass.value) {
        plus.nativeUI.toast('密码两次输入不一致');
        return false;
    }
    if(!meishiApp.regExp.notEmptyReg.test(reg.email.value)) {
        plus.nativeUI.toast('请输入 Email');
        return false;
    }
    if(!meishiApp.regExp.emailReg.test(reg.email.value)) {
        plus.nativeUI.toast('请输入正确的邮箱');
        return false;
    }
    if(!meishiApp.regExp.notEmptyReg.test(reg.tel.value)) {
        plus.nativeUI.toast('请输入手机');
        return false;
    }
    if(!meishiApp.regExp.mobileReg.test(reg.tel.value)) {
        plus.nativeUI.toast('请输入正确的手机号');
        return false;
    }
    return true;
}
```


14.9 我的订单

我的订单(myorders.html)窗口需要单击首页中“我的”选项卡,单击“我的订单”后打开。窗口中会列出固定的 5 条订单记录,如图 14-36 所示,并显示订单的评价状态为“已评价”或“待评价”。



图 14-36 “我的订单”窗口

14.9.1 artTemplate 生成列表

在所有的 HTML5 App 开发中,或多或少都会遇到将数据生成界面上相应列表的问题。通常的做法是使用 DOM 动态生成列表的各标签对象和属性,这种做法的程序可读性较差,而且一旦需要修改界面时,代码维护相对比较困难,美食详情中的“评论列表”功能就是用这个方法实现的,可以参看 getcomment.js 中的代码。这里使用了另外一种技术——腾讯的 artTemplate(<https://github.com/aui/artTemplate>),它是一种 JavaScript 模板引擎,性能卓越,语法简洁,支持所有流行的浏览器,使用时需要“template.js”支持。

参看如图 14-36 所示的效果,我们在“我的订单”窗口上,先按静态页面设计的 HTML 代码设计出这个列表项的模板,代码如下:

```
<!-- 定义订单列表项的模板 -->
<script type="text/html" id="oList">
```

```

<li class="mui-table-view-cell mui-media"
      data-id="{{OrderID}}">
  <a href="javascript:;">
    
    <div class="mui-media-body">
      {{OrderTitle}}
      {{IsCommented}}
      <pclass="mui-ellipsis">数量:&nbsp;{{OrderCount}}</p>
      <span></span>
      <p class="mui-ellipsis">总价:&nbsp;¥ {{OrderAmount}}</p>
      <span></span>
    </div>
    {{if IsCommented}}
      <div class="comment_area">
        <a href="javascript:;">已评价</a>
      </div>
    {{else}}
      <div class="comment_area">
        <a href="javascript:;">待评价</a>
        <button class="comment">评价</button>
      </div>
    {{/if}}
  </a>
</li>
</script>

```

其中,类似于“{{OrderCount}}”这样的语法,表示将 JSON 对象的 OrderCount 属性显示在该处。artTemplate 也支持条件判断,例如像上面 {{if IsCommented}} 这种语法,表示当 IsCommented 这个属性为 true 时,完成“已评价”的 HTML 内容生成,{{else}}则表示否则生成“待评价”的 HTML 内容。

模板语法可读性要好得多,更为重要的是维护难度并不大,当取得数据后,只需用下面的代码,将数据与模板结合,生成相应的 HTML 代码:

```

//按模板生成需要的 html,res 是 JSON 对象数组
var html = "";
for(var i = 0; i < res.length; i++) {
  html += template("oList", res[i]);
}
myorders.oul.innerHTML = html;

```

对于模板需要复用的地方,例如美食列表和收藏的美食列表需要使用的是同一个模板,我们采用的方式是定义一个模板文件“template/food. tpl”,然后在窗口打开后,采用 AJAX 的同步加载功能加载模板。在“App.js”中定义了加载模板的方法。

```

/* 获取相应的 artTemplate 模板,采用同步获取 */
meishiApp.getTemplate = function(tmpl, container) {

```

```
mui.ajax(tmpl, {
    type: "GET",
    async: false,
    success: function(res) {
        container.innerText = res;
    },
    error: function(xhr, type, errorThrown) {
        //异常处理;
        console.log(type);
    }
});
}
```



同步的方式意味着必须加载完模板之后才能执行后面的 JavaScript 代码。由于模板会被 APK 打包在本地，所以效率不会差。

14.9.2 评论

对于“待评价”的订单，单击“评价”按钮，会进入评价窗口（feedback.html），在这里可以输入评语，也可以选择快捷输入，从相册选择图片或拍照，如图 14-37 所示，进行星级评分后发布评论。这个窗口的页面也是使用 HBuilder 的内置模板创建后修改的，如图 14-38 所示。

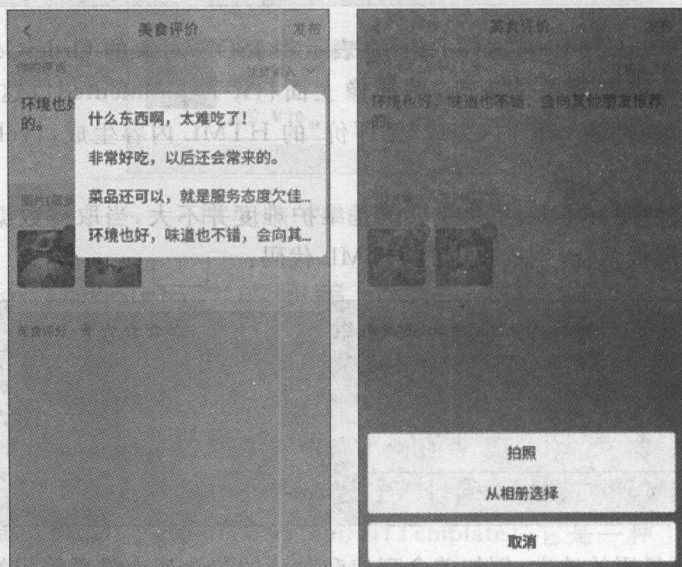


图 14-37 “美食评价”窗口

在窗口中，单击图片添加按钮时，窗口底部弹出的 actionsheet 是采用的原生效果，基于 HTML5+ 规范的 NativeUI 模块，代码如下：

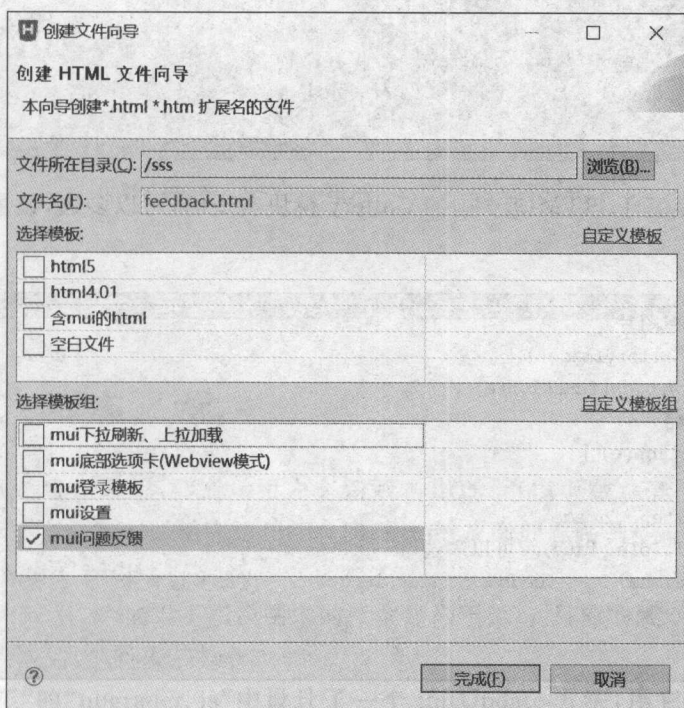


图 14-38 “mui 问题反馈”模板

```

plus.nativeUI.actionSheet({ cancel: "取消",
  buttons: [{ title: "拍照" }, { title: "从相册选择" }] },
function(e) {
  if(e.index == 1) {
    //处理拍照
    dealCameraImage(placeholder);
  } else if(e.index == 2) {
    //处理相册选择
    dealGalleryImage();
  }
});

```

拍照是基于 HTML5+规范的 Camera 模块实现的,代码如下:

```

var cam = plus.camera.getCamera();
cam.captureImage(function(e) {
  plus.io.resolveLocalFileSystemURL(e, function(entry) {
    var s = entry.toLocalURL();
    //代码略 ...
  }, function(e) {
    console.log("读取拍照文件错误:" + e.message);
  });
}, function(e) {
  console.log("Camera error:" + e);
});

```

```

    }, {
        filename: "_doc/camera/", //保存路径
        index: 1 //表示主摄像头
    })

```

相片选取则是基于 HTML5+ 中的 Gallery 模块实现的,可以多选,数量不超过 9 张,核心代码如下:

```

//处理相册选择图片
function dealGalleryImage() {
    plus.gallery.pick(function(e) {
        //代码略...
    }, function(e) {
    }, {
        filter: "image",
        multiple: true, //可以多选
        maximum: 9 - feedback.files.length //一次最多选 9 张
    });
}

```

不论是拍照还是从相册中选择图片,如果直接在页面上显示图片,效果都会很差,因为图片显示缓慢,而且内存占用较高,所以选择或拍完照片后,都要使用 HTML5+ 的 Zip 模块对图片进行压缩,以 3M 大小的图片为例,压缩后只有约 500KB。在页面上显示的实际是压缩后的图片路径,代码如下:

```

//压缩图片,减小文件尺寸
plus.zip.compressImage({
    src: img, //源路径
    dst: "_doc/zip/" + name, //压缩后的路径
    overwrite: true, //可以覆盖
    quality: 50 //压缩质量为 50 %
}, function(zip) {
    //显示图片
    placeholder.style.backgroundImage = 'url(' + zip.target + ')';
}, function(e) {
    console.log("压缩失败了!");
});

```

发布评论时,实际上先通过指定的 API 上传图片,得到图片存储的 URL 地址后,再和评论内容与星级评分一起通过 API 保存。上传图片是利用 HTML5+ 的 Uploader 模块,核心代码如下:

```

var uploader = plus.uploader.createUpload(api_url.uploadingUrl,
    {method: "POST"}, function(upload, status) {
        //所有文件都上传成功了
    }
);

```



```

        if(upload.state == 4 && status == 200) { //代码略... }
    });
    //依次将图片添加到上传任务中
    for(var i = 0; i < feedback.files.length; i++) {
        //上传任务添加
        uploader.addFile(feedback.files[i].path, {key: Math.random()});
    }
    //开始上传
    uploader.start();

```

14.10 版本更新

现在的 App 在启动时,会提示用户有新版本出现,可以下载最新版本的 App 并安装。我们在“美食汇”中也实现了相应的功能。实现 App 更新的思路基本都是类似的:

- 下载 App 更新说明;
- 比较 App 版本,如果更新说明中的版本号高于 App 目前的版本号,则进行下载;
- 下载完成后,执行安装操作。

在“美食汇”的“upgrader.js”中设计了一个 initUpdate 方法,用它来完成 App 的更新操作。

首先需要在服务器端提供更新说明文件,它的内容实际是一个 JSON 对象定义,“美食汇”的更新说明文件的地址是: <http://www.meishihui68.com.cn/update.json>,它的内容如下:

```

{
    "Appid": "美食汇",
    "Android": {
        "version": "2.0",
        "title": "美食汇版本更新",
        "note": "新增分享功能\n修正了一些错误\n",
        "url": "http://d.m3w.cn/helloh5p/HelloH5.apk" //新版本下载地址
    }
}

```

在设置窗口(settings.html)中单击“版本更新”后,我们使用 MUI 的 AJAX 技术向服务器请求,可以轻松拿到这个说明文件的内容,从中取出 version 这个属性,用它与当前的版本号进行比较,如果高于当前版本号,则打开自定义升级窗口(upgrade.html)进行升级,效果如图 14-39 所示。

使用 HTML5+ 的 Runtime 模块取出当前 App 的版本号,代码如下:

```

var curVer = null; //当前版本号
function initUpdate() {
    //取当前 App 的版本号

```

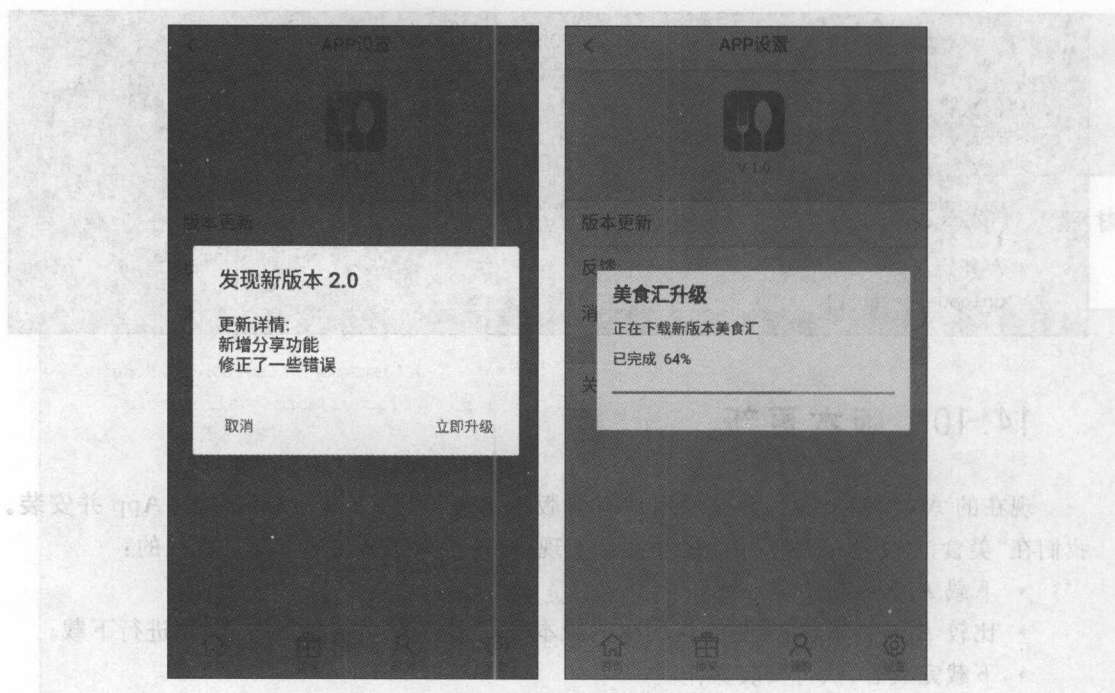



图 14-39 App 版本更新升级效果

```
plus.runtime.getProperty(plus.runtime.Appid, function(inf) {
    curVer = inf.version;
});
}
```

当前 App 的版本号比说明文件的版本低, 在打开升级窗口后, 使用 HTML5+ 规范中的 Downloader 模块, 从新版本安装包的 URL 地址开始下载, 代码如下:

```
var dw = plus.downloader.createDownload(updateDownloadUrl, {
    filename: "_doc/update/", //下载的目录
    timeout: 30
},
function(d, status) {
    if(status == 200) {
        //安装下载的 APK 包
        installUpdate(d.filename);
    }
});
dw.addEventListener("statechanged", onStateChanged, false);
dw.start();
```

在下载过程中, 需要使用状态监听函数实现进度条, onStateChanged 的定义如下:

```
function onStateChanged(dw, status) {
    if(dw) {
```

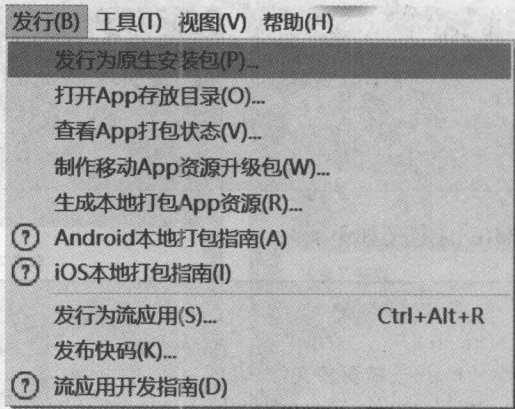



图 14-41 发行为原生安装包

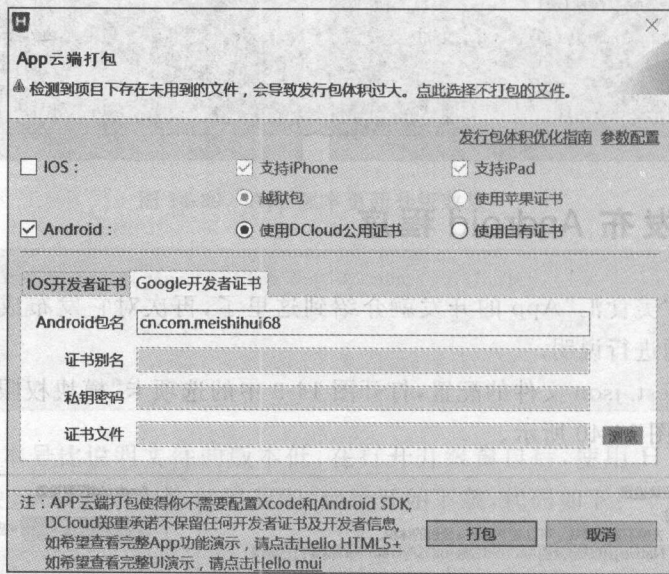


图 14-42 “App 云端打包”对话框

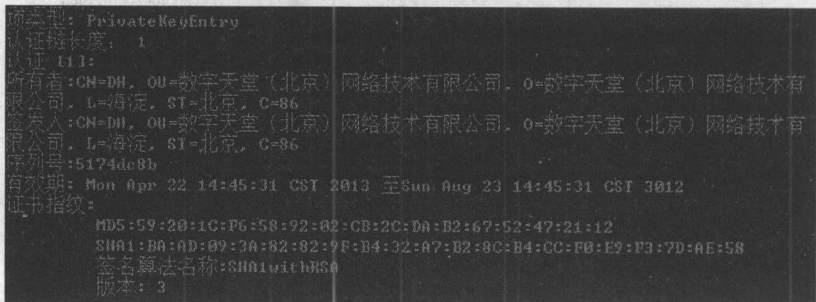


图 14-43 DCloud 公用证书信息

DCloud 的公用证书的相关信息如图 14-41 所示,也可以使用自有的证书,关于 Android 证书的生成可以参看 <http://ask.dcloud.net.cn/article/191>。如果想实现自己的离线打

部分习题参考答案

第1章

一、1. C 2. ABD 3. D 4. B 5. AB

二、× × × × ×

三、1. MUI 2. HTML5+

第2章

一、1. B 2. D 3. D 4. B 5. B 6. B 7. B 8. B 9. D 10. B

二、√ × × × √ √ ×

三、1. <body></body> 2. <select><option> 3. placeholder 4. <table>

5. <! --注释语句-->

四、1. 见书籍第30页。

2. HTML5 标准制定了许多语义化标签,它可以将每个区域语义化,让页面结构更清晰,更利于 SEO。

第3章

一、1. C 2. A 3. B 4. A 5. C 6. C 7. C 8. C

二、√ × × √ √ ×

三、1. 内联样式 内嵌样式 链接样式 2. # 3. 依据最近已经实现过定位(相对、绝对、固定)的父元素进行定位,若所有父元素都没有定位,则根据 body 元素(浏览器窗口)定位 4. float:left; clear:both;

五、(见配套资源包)

第4章

一、1. B 2. A 3. A 4. A 5. A

二、× × × × ×

三、1. var 2. === 3. function 4. setInterval setTimeout

四、2. this 总是指向调用该方法的对象

第5章

一、1. B 2. B 3. B 4. C 5. D

二、√ × √ × ×

三、1. Document Object Model 文档对象模型 2. 查找 增加 修改 删除 复制

3. 新增 删除 修改 4. 捕获阶段 目标阶段 冒泡阶段 5. 事件参数

五、(见配套资源包)

第6章

一、1. D 2. C 3. C 4. C 5. C 6. D 7. D 8. D 9. B 10. B

二、× × √ × ×

三、1. \$("input") 2. each() 3. innerText 4. \$() 5. one() 6. \$.extend()

7. true 8. prependTo()

五、(见配套资源包)

第7章

一、1. B 2. C 3. B 4. C 5. D 6. B 7. A 8. B 9. D 10. B

二、× × × × √

三、1. Asynchronous JavaScript and XML

2. Content-Type: text/xml Content-Type: application/xml 3. 404 4. 请求 响应
无状态

5. dataType json 6. DELETE

7. Access-Control-Allow-Origin Access-Control-Allow-Headers Access-Control-Allow-Methods

五、(见配套资源包)

第8章

一、1. A 2. D 3. C

二、× × × × ×

三、1. 轮询 2. Sec-WebSocket-Accept 3. WebSocket

第9章

一、1. D 2. C 3. C 4. B 5. A

二、× √ √ ×

三、1. <audio><video> 2. 用来定义视频播放器的预览图片 3. controls

4. playbackRate 5. waiting

第10章

一、1. B 2. C 3. B 4. D 5. D

二、√ × × √ ×

三、1. NoSQL 2. removeItem 3. 关闭了页面或浏览器

第11章

一、1. B 2. A 3. C 4. A 5. B 6. D 7. B 8. C

二、× × √ √ × ×

三、1. arc 2. createLinearGradient createRadialGradient 3. 4 4. moveTo

5. lineWidth

四、(见配套资源包)

第12章

一、1. B 2. D 3. A 4. D 5. D

二、√ × × √ √

第13章

一、1. C 2. B 3. B 4. B 5. C

二、× √ √ √ √

三、1. mui.min.js mui.min.css 2. mui.os.ios 3. tap 4. mui.openWindow

5. back

五、(见配套资源包)

参考文献

- [1] 潘中强,曹卉编.构建跨平台 APP:HTML5+PhoneGap 移动应用实战.北京:清华大学出版社,2015.
- [2] 传智播客高教产品研发部.HTML5+CSS3 网站设计基础教程.北京:人民邮电出版社,2016.
- [3] 李雯,李洪发.HTML5 程序设计基础教程.北京:人民邮电出版社,2016.
- [4] DCloud 公司官方文档. <http://dcloud.io/doc.html>

HTML5 App Design and Development

HTML5 App

应用开发教程

本书介绍HTML5在移动App开发领域的相关技术、CSS3的应用、JavaScript的编程技术，并使用大量实例介绍了利用Hbuilder、MUI、HTML5+规范开发App的流程和实现。

本书特点

契合认知规律

从基础、实用的内容出发，由浅入深，内容丰富。读者在认真学习之后，能够基本具备HTML5 App的开发能力。

注重阅读体验

讲述每个知识点的同时，给出了丰富的插图与完整的实例。书中实例都来源于实际开发项目，便于学习。同时还讲述了很多实用技巧与开发心得，具有较高的参考价值。在最后一章给出了一个App开发的综合实例。

丰富学习资源

提供学习建议、演示文件（PPT）、实例源代码、练习文件、习题等。下载地址为清华大学出版社网站本书页面。书中实例所需要的服务端API已进行了Internet部署，可以直接调用。

课件下载·样书申请



书圈

清华社官方微信号



扫我有惊喜

上架指导：计算机/HTML5

ISBN 978-7-302-48199-7



9 787302 481997 >

定价：79.00元